UNIVERSITY OF OSLO
Department of Informatics

Open web based evaluations at UiO

An experiment with the web application kurskritikk.no

Master thesis

Daniel Chaibi

May 28, 2008

Preface

The last 5 years I have attended the Masters Degree Program at the Institute of Informatics at the University of Oslo. During these years my interest for web technology and related topics has increased. The spring 2007 I took the course INF5270 (Design of interactive web sites) where an idea of mine about online open course evaluations was partially realised in a project course. We built a social web site for course evaluations, but the project never reached our goals in functionality and was abandoned after delivery. As the idea behind the project was important to me, I decided to give it a new go in my master thesis.

The choice of topic is driven a mix of my interest in the field of web technology and an observed need for renewal of course evaluations at the University of Oslo. As the thesis was formed, the passion for the site grew, and I decided to launch the whole idea at the URL www.kurskritikk.no. I will administer and building new functions on the site on a hobby basis after delivery of the thesis.

Wikipedia is used as a source in some sections of this thesis. I am aware of its lack of credibility in some environments, but in consultation with my supervisor I have chosen to use it as source for some subjects. The quality of the content is varying, but my experience with technical subjects has been good. I never use it without confirming the content against a couple of other sources.

A helpful resource has been my supervisor Gisle Hannemyr which has special interest in the theory behind this work. I would like to thank Fredrik Fjeld and Jon Lønne for great assistance with Django, all the people from the irc-group prosit for both great professional and social discussions. Thanks to all students who have contributed with evaluations while I wrote the thesis. Please keep it up so many students after you can enjoy and benefit from it.

Last but not least I want to thank my family and friends for keeping my spirit up this last year. Special thanks to Anette who has been my fellow student for many long days and nights ... we finally made it!

Abstract

Social websites have had an enormous growth the last couple of years. One of the success factors has been user contributions and great use of meta data on these contributions. While the developers can concentrate on functionality, the amount of content grows by itself. The fact that the content is unfiltered raises some interesting challenges. With big amounts of content with variable quality, it is hard to find what is most useful for the users. This is where a quality measurement has to be done. Especially in review sites where users are looking for comments and meanings on various elements (eg. consumer products like books, music, gadgets etc.).

In this thesis the theory behind this is used to build an example system for user contributed evaluations of courses using the Python based web framework Django A algorithm for quality measuring evaluations is proposed and tested on 71 evaluations from kurskritikk.no. The algorithm includes two main variables where the first is meta-data (thumbs up/down on usefulness) and the second text properties of the evaluation. The thesis has resulted in a full functional web site for open online course evaluations and the algorithm works well in quality measurement of the evaluations. It is not complete, but is built in a way that makes future expansions easy to implement.

Contents

Chapter 1

Introduction

The University of Oslo, as many others, has an evaluation system for their courses. Students can use this to give feedback on specific courses and their quality. The information is being used to help the course administration to constant improve the overall quality of each course.

The feedback is only made available for the course administrations, and this is where I feel huge improvements can be made. Not only does all this feedback have a great value for the students who give them, but it can also be used to generate new and interesting information. Aggregating data like this makes it easier to look at the information from new angles, both for administrations and students. By giving evaluations in an open social website, the administrations keep their insight at the same time as students can take advantage of the same information. By giving students immediate value of their contributions, it may also increase their participation.

1.1 Goal of this thesis

My goal is to make a full functional site for course evaluations, from here on called 'Kurskritikk'. In the theory part I will focus on online identities, reputation, credibility and quality measure of online content. The goal is not to replace today's system, but make an open alternative. Much of the work in this thesis is done developing the web application Kurskritikk.

1.1.1 Research question

Making this open alternative generates certain challenges, and this is where the theory and research side of this thesis finds place. For a site like this to function as good as possible it has to balance a couple of elements. This is what I aim to provide through Kurskritikk:

- Anonymity for all users.

- Available and credible evaluations.
- Possibility to generate key information by aggregating data.

This must of course build on a steady and easy to use platform, hence some technical goals as well:

- Appealing and good user interface.
- Low threshold for use.
- Cross browser compatibility.
- A constant updated course database reading UiO's public available XML files.
- Secure database ensuring the users anonymity.

How I am going to achieve these goals is what this thesis will discuss together with the implementation of the actual system. Traditional trust is hard to combine with anonymity, and here is where I find many interesting challenges. I will focus on what is stated in the following problem formulation:

> *How can we obtain high quality and credible information from online users based on full anonymity?*

1.1.2 Technology platform

Building web applications from scratch is often a comprehensive and cumbersome task involving a lot of debugging and cross-browser compatibility issues. In order to keep focus on the functions and theory of the thesis I made some important choices. First of all I chose the Python web framework Django. This framework does a lot of the business logic for you and eases tasks such as URL handling, templates and models. The framework in its self is cross browser/os compatible, so what I need to focus on to keep it that way is writing CSS, HTML and JavaScript which works well across all operating systems and browsers. With this as a basis everything should be set to build a good web application. More on the technical parts in chapter 4.

1.2 Outline of thesis

During the next chapters I will build a theoretical foundation for the reader, go through some of today's systems before I describe how Kurskritikk is built and shows some test results and screenshots of the web application. A listed outline of the thesis follows.

Chapter 2 goes through the background and theory of the thesis.

Chapter 3 analyses existing web applications with special relevance of Kurskritikk.

Chapter 4 describes how Kurskritikk is built and thoroughly examines the quality sorting algorithm developed for the system.

Chapter 5 discusses the choices made in the application and analyses the results of the sorting algorithm.

Chapter 6 concludes the thesis.

Appendix A contains a systems requirements spesification, including all imporant use cases.

Appendix B contains a documentation of the system, including how the web framework is used.

Appendix C contains some attachments.

Chapter 2

Theory

This section will cover all the theory connected to the system being developed in this thesis. As my research question stated, I want to obtain high quality information from online users while still giving them full anonymity. This touches some interesting topics that have to be covered before we go into any further details on how this can be achieved.

2.1 Social network service

Though I will not build any social networking into Kurskritikk, I still want to go through the theory of it since it is very relevant for the thesis' broader field of topic. It is possible though, that some elements of social networking can be built in to Kurskritikk at a later stage.

Wikipedia defines social network service like this [18]:

> "*A social network service uses software to build online social networks for communities of people who share interests and activities or who are interested in exploring the interests and activities of others.*"

The last 5 years has shown a huge increase in both social network services and users of such services. It has become the new playground for all kind of people to socialize and share bits and pieces of themselves and their networks. These services are primarily web-based and the biggest of them (Facebook[1], MySpace[2] and Orkut[3]) has tens of millions of users which use these sites as a part of everyday life.

Social network services can be broken down into two categories, internal social networking (ISN) and external social networking (ESN). An ESN is an open/public community where everyone can join and communicate with each other. This is where Facebook and

[1] http://www.facebook.com/
[2] http://www.myspace.com/
[3] http://www.orkut.com/

MySpace are categorized. An ISN is a closed/private community consisting of people within a specific group (company, society, education provider etc.). An ISN can also exist in an ESN in the form of an "invite only" group.

It started as early as 1995 with the service Classmates[4] which intended to connect former classmates. It had simple features like messages, friend-lists and interests, which could be used to find people with similar interests to you. This has now expanded greatly and features on today's social sites are now growing almost as fast as the list of users on these sites. The crown example is Facebook which in 2007 allowed externally made add-on applications which builds on the whole social networking idea. Not only do the users provide the content, they also provide the functionality.

2.1.1 Social networks in more domains

As it may seem, social networks is not only used to connect friends and share photos, it is more to it than that. Businesses use it to connect to and share ideas and knowledge with other colleagues, where LinkedIn[5] (see section 3.3) is the major player with over 20 million users from 150 different industries.

Another business domain is healthcare, where social networks are used to manage institutional knowledge and stimulate to share and develop knowledge in the field. An example is Sermo[6], which reminds some of LinkedIn. The difference is that Sermo is only open for physicians and will therefore keep focus on pure medical discussions. They also verify that the users are licensed physicians practicing in the United States.

Another use of social network services is dating services. This is communities where people register with their preferences in other people with the goal of being matched with people interested in you. Examples of such services are match.com and Yahoo! Personals[7].

Social networks exist in many more specialised domains such as travel and lifestyle, photos, sports, games etc. Social network aggregation is a new concept for connecting these networks together. Many users are member of several online communities, making a lot of overlap [13]. Two concrete initiatives are OpenID[8] for cross-site user IDs and OpenSocial[9] for common APIs for these social network applications.

2.2 Trust

An important attribute in Kurskritikk is trust. The main focus in Kurskritikk is to make today's course critics useful for students as well as the course administration. In order to achieve that, the information given by users has to be trusted.

[4] http://www.classmates.com/
[5] http://www.linkedin.com/
[6] http://www.sermo.com/
[7] http://personals.yahoo.com/
[8] http://www.openid.net/
[9] http://code.google.com/apis/opensocial/

Before we go into detail in which kind of trust is needed in Kurskritikk, I want to proceed with a definition trust.

Sztompka (1999) [16] presents a simple definition of trust. Sztompka says "*Trust is a bet about the future contingent actions of others.*". This definition is divided into belief and commitment. A person will only trust somebody fully when s/he commits a belief into a given action.

In order to execute this belief, trust has to be built. Trust is built through a history between two agents. An agent can be everything from a person, a brand to an application. Common for all of them is that humans always stand behind their behaviour. "*All human actions occur in time, drawing upon a past which cannot be undone and facing a future that cannot be known*" (Barbalet 1996 [1]). The history can either be negative or positive, and of course something in between. The more positive a history is, the higher is the chance to gain trust. When people share this trust history with each other, they build what we know as reputation.

As Barbalet mentions, we cannot know the future. In order to gain trust, we have to make some bets about the future. The outcome of these bets determines how much we trust an agent. Negative outcomes often counts more than positive ones. Ebay is a good example of that. Let's say you want to buy a $1000 camera from the user with the alias CameraSeller. CameraSeller have received 20 feedbacks of which 4 are negative, consequently 80% of them positive. You will most certainly hesitate buying from this person, especially if the negative feedbacks are about some kind of fraud. The 16 positive feedbacks are placed in the shadow of the 4 negative ones, which builds on Barbalets second fact; the past cannot be undone.

Each person has a circle of trust (see figure 2.1 on the following page), where the centre contains the most trusted. Family is often placed here as the people we trust the most, this is natural as we spend most of our childhood and youth with them. The further out from the centre we go, the less we trust the subjects. Building trust involves two factors, (1) the trusted and trustee(s) and (2) the topic of trust. The people closest to your inner circle are people you trust and tell your secrets and intimate details about your life to. This can be family and close friends. They can also be helpful when you are in the need of useful opinions on e.g. clothes, restaurants and movies.

A step further out in the circle we find experts and people with the experience you need and can take advantage of. Online shoppers are getting better at using such resources on the Internet. One of these is expert reviews on nearly anything. Which of them you decide to trust depends on the history of each of them, their reputation. This reputation can be on a magazine publishing a review, or directly on the reviewer. On community sites like Amazon.com, you have to trust the users directly and through user provided usefulness scores on the reviews. You will then trust the community as a whole. This is where Kurskritikk's place in the circle is.

This brings me over to a key aspect of trust on the Internet, the *Web of Trust* (WOT). We can look at it as a circle of trust with a little twist. The circle of trust contains agents you, and only you, trust directly and indirectly at different degrees. The web of trust on the other hand is a web containing the trust statements for all agents in a

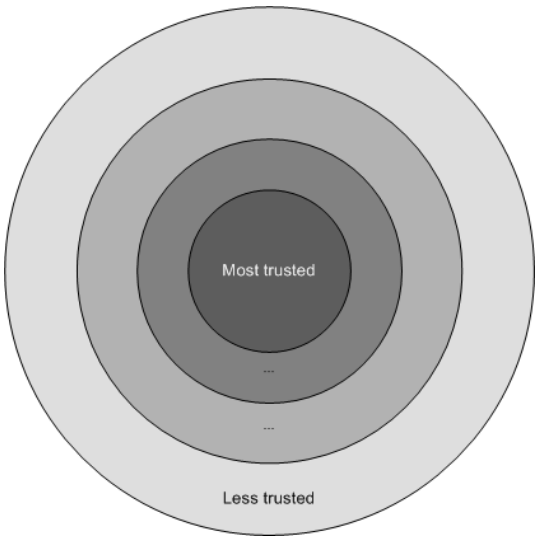

Figure 2.1: Circle of Trust

network. Each agent is a node and edges between them describe their trust relationship which can be either positive or negative. No edge between two agents indicates no relationship (see figure 2.2 on the next page). Trust management systems do not need edges between two agents to build a trust relationship. If Agent A trust B, and B trust C, then a system could use this to imply that A trust C. The more trust statements a system gets from its users, the more useful the WOT gets.

2.2.1 Reputation management

Trust as explained above is often obtained by some form of reputation. Wikipedia's definition of reputation says;

> "*Reputation is the opinion (more technically, a social evaluation) of the public toward a person, a group of people, or an organization. It is an important factor in many fields, such as business, online communities or social status.*"

In addition to people and organizations, a reputation can also be built around an agent, a product or a service.

While trust is personalized and subjective reflecting an individual's opinion on an entity, reputation relies on the aggregation of every individual/agents experience with it. Reputation can be inherited, for example a brand, let us say Volvo who sells cars has a good reputation on almost all of their models. They can release a new model which will inherit Volvos good reputation and be easier to sell than a new model from a new brand. Regardless of inheritance a good reputation takes time to build and involves many people/agents.

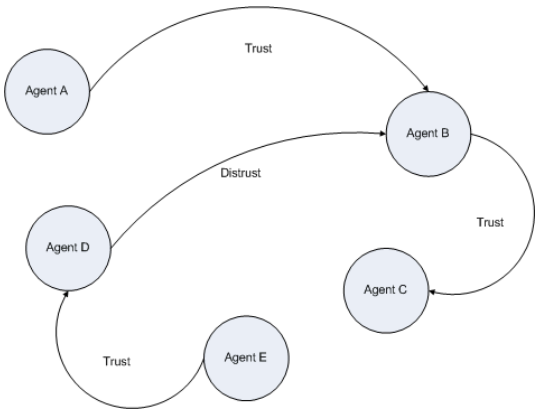

Figure 2.2: Web of Trust

Reputations have existed since long before the digital age, but reputation management is a concept which grew in size and use when computers started to be used widespread. Wikipedia defines it like this;

> *"Reputation management is the process of tracking an entity's actions and other entities' opinions about those actions; reporting on those actions and opinions; and reacting to that report creating a feedback loop. All entities involved are generally people, but that need not always be the case. Other examples of entities include animals, businesses, or even locations or materials. The tracking and reporting may range from word-of-mouth to statistical analysis of thousands of data points."*

Wang and Vassileva [17] uses three criteria's to analyze and classify today's trust and reputation systems. Before I summarize them, I will present three common characteristics of trust and reputation:

- **Context specific**

  Trust and reputation depends on some sort of context. You can trust John to fix your computer, but not as a pilot of your airplane.

- **Multi-faceted**

  context-specificity refers to trust and reputations that varies in different situations while multi-faceted emphasizes that a single situation can have multiple aspects. An example can be an online service where different Quality of Service (QoS) aspects such as response time, accuracy, execution time and security can decide whether the service is trustworthy or not.

- **Dynamic**

  Trust and reputation increase and decrease over time as further experiences and observations are gathered. New experiences are more important and relevant then old ones, and very old ones can become obsolete or irrelevant.

As we can see there are a couple of aspects involved in trust and reputation. When building such systems it is important to be aware of all these aspects, and even more importantly decide which ones that is important for a specific system.

There are a whole range of trust and reputation systems out there today (see figure 2.3 on the facing page). In order to see how we can build and maintain systems of high quality and relevance, we have to be able to classify them. Figure 1 shows a three-level hierarchy classification of today's systems, and I will now go through their definitions.

- **Centralized vs. decentralized**

  At the top of the hierarchy we can choose a centralized system where a central node will take the responsibility of managing the members reputations. In decentralized systems there is no central node and users have to co-operate and share the responsibility in order to manage the reputation. Centralized systems are less complex than decentralized ones, but they require powerful and reliable central servers with a lot of bandwidth for computing, data storage and communication.

- **Person/agent vs. resource**

  A trust and reputation system can be classified as a person/agent where the reputation is modelled directly on people or agents. With a resource system the reputation is built around resources such as products and services. Some systems such as Amazon[10] and Epinions[11] build reputations around people/agents with the purpose of building a reputation on the resources. Ebay on the other hand focuses on the persons and only their reputation.

- **Global vs. personalized**

  In a global system the reputation of an entity (person, agent, product or service) is based on the general population and is public and visible to all members. In personalized systems the reputation of an entity is built from a limited group of people. This group may vary and be influenced by many factors such as members' social networks. As an example in Kurskritikk we can offer the students personalized recommendations based on reputations on users with the same course combinations as the student, or other factors such as age, institute and so on.

2.3 Privacy

Privacy is all about controlling people's access to you, your possessions and information about you. A definition from Wikipedia describes the attributes of privacy in a short and concise way;

> "*Privacy is the ability of an individual or group to seclude themselves or information about themselves and thereby reveal themselves selectively*".

[10] http://www.amazon.com/
[11] http://www.epinions.com/

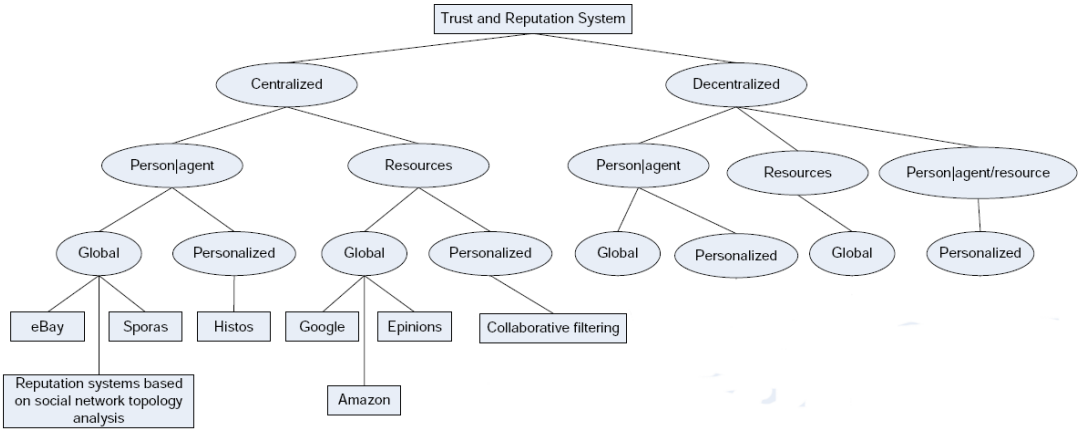

Figure 2.3: Trust and reputation system classification (from [17])

There are many ways of dividing privacy into groups. There have been written articles and books about privacy, but it is beyond the scope of this paper to cover all aspects of privacy. Privacy International has given a nice division of the aspects of privacy [8];

1. **Information privacy**, which involves the establishment of rules governing the collection and handling of personal data such as credit information, and medical and government records. It is also known as 'data protection'.
2. **Bodily privacy**, which concerns the protection of people's physical selves against invasive procedures such as genetic tests, drug testing and cavity searches.
3. **Privacy of communications**, which covers the security and privacy of mail, telephones, e-mail and other forms of communication.
4. **Territorial privacy**, which concerns the setting of limits on intrusion into the domestic and other environments such as the workplace or public space. This includes searches, video surveillance and ID checks.

1 and 3 is the aspects on the web with most relevance. As people spend much of their time online, territorial privacy gets a new dimension. With applications such as the virtual world Second Life[12] and the game World of Warcraft[13], people get ownership of new territory. Privacy applies in real life as well as in the mention fiction games. IBM has even built an office [7] in Second Life where real people from their staff is working through an avatar. In Kurskritikk we meet mostly information privacy, but also some privacy of communications.

In Norwegian universities the lecturers and other employees does not have direct absolute power to determine the grades of students. This is due to the censoring system

[12] http://www.secondlife.com
[13] www.worldofwarcraft.com

we use where students only write a random number, not names on their exams. This censoring system is used on most, but not all courses. Some courses do have names attached to exams and assignments, and we also have the oral examinations where names are used. This taken into consideration, a site like Kurskritikk needs to preserve the privacy of its users. As it is now, their real identities could, but most probably not, be used against them in grading if they are not anonymous in evaluations.

2.3.1 How to ensure users privacy

The privacy of users can be ensured in various ways, at different degrees and at many levels in the system. One way is to offer users anonymity in the front end (what the other users see) and/or in the back-end (what the system sees). In the front-end we can hide the identity of the author behind an evaluation or we can hide them behind an alias. With an alias a user can be identified in the case of rare course combinations which only apply to a small group of students. In the back-end we can hide the author by eliminating the link between user and evaluation. This can be done by adding evaluations to a separate table in the database without any foreign keys to the users. I will now go in more depth on some means of ensuring users privacy.

Anonymity

Anonymity most often refers to a person which means that the identity or identifiable information about that person is not known. On the Internet anonymity is often used where users contribute with their opinions in some kind of forum. This can be discussion forums, review sites and comments on blogs. Users tend to feel safer when writing anonymously or pseudonymously (section 2.3.1 on the next page). In general we want to be anonymous when what we do or say can be used against us when linked with our real identities.

Anonymity can be ensured at different levels in a system. At the bottom we have what we call connection anonymity, which is about hiding the identities of source and destination during the actual data transfer [4]. This is out of the scope of this paper, and what I will focus on is data anonymity, which is about filtering any identifying Information out of the data.

In general we achieve true anonymity when the content (in my case the evaluations) are not linked to the user in any way, a concept called unlinkability. As default Kurskritikk will provide this in the front end. The evaluations will be unlinkable to the users in the front end. In the database the evaluations will be connected to the users who wrote them. This is because I want the data to be available to build functions such as a WOT. This can be used to recommend evaluations which are extra relevant for a user. If an attacker gets access to a system where content is unlinked to the author, there will be no way of extracting who wrote which evaluations.

Online content from anonymous users (reviews, discussion forums, etc.) has proven to give various results. In general, users are less afraid to write negative, irrational and

sometimes direct mocking text on such sites. In 2005 Kilner and Hoadley got access to data from an online community of practice (CoP) for U.S. soldiers [10]. The CoP was in a changing state where it went from total anonymous users, through aliases and finally to users with full names. Kilner and Hoadley analyzed posts from all three identity states and came to some interesting but not surprising results. They gathered posts from equally sized time frames for each identity states. When users could be anonymous they had 707 posts, where 435 were anonymous. 11% of them was negative. In the alias state the participation decreased by 321 and the percentage of negative posts fell to 2%. An interesting result was that moving from alias to full names did not improve the seriousness as much. This could indicate that people do not feel they can hide between an alias, thus, showing more seriousness in their writing.

Alias

An alias is used when a user do not want to disclose their real identities to others, but still be able to build a visible reputation. The alias is chosen by the user and identifies one or more holders. It is used when the holder do not want to disclose their true names. Aliases are most often used when the holder want to remain anonymous, but still be able to build a reputation. Examples of this are Ebay and Slashdot. In Ebay the user's needs a persistent alias that sellers and buyers can give a rating after a closed deal. In Slashdot the pseudonym holder builds a reputation around the content s/he publishes, and a better reputation gives more visible content. This gives the holder a feeling of accomplishment and the readers get higher quality content more visible.

Aliases will never give true anonymity. In some cases the holders have a loose alias where a human can potentially link the alias to a real identity. This could be due to the fact that parts of the name are included in the alias. If it is not linkable in the front end, there will most often be a direct link from the alias to the identifiable information the user has provided in the back-end. This could be e-mail address, real name or phone number. Another danger is fingerprinting, which is when readers use the content which can be unique for a users, to pin down the authors identity. In Kurskritikk this can be users who have completed a unique or almost unique set of courses which is known to one or more people.

Pseudonymity

Pseudonymity is in many cases confused with aliases. Wikipedia is among the sources that does that. A pseudonym is a machine generated identifier used as a persistent link between user generated content and the author. A pseudonym should be used when we want to keep the users anonymous, but still have a persistent link on the content. This link could be used to build webs of trust.

The pseudonym identifier should not be, in the normal course of events, sufficient to associate the transaction with a particular human being [2]. In order to achieve this, a complex infrastructure has to be built around the generation and usage of these pseudonyms. I will not go into any details on how to implement this, as that is out

of the scope of this thesis. The overall principle is that a third party would have to be used. The content owner, let's say Kurskritikk, would then at registration of a user communicate with a third party of disputed integrity and have a persistent pseudonym generated. There would not be enough data in either of the systems to link the pseudonym to the possible identifiable information the user is registered with. Some sort of cryptation should be used so that not even the owners of either system could connect a pseudonym to an identity. If one of the systems gets compromised, the identity of the user would still be a secret. Only if both systems are compromised, the identity can be found.

The user could also be its own third party if s/he registers with an alias that is only known to him/her. No other potential identifiable information, including e-mail, must be stored, and this alias must not be visible in the front-end. The drawback is that the user has to remember the alias as no e-mail can be stored and used to send out password reminders/resets.

2.4 Quality measuring evaluations

The online course reviews that I want to measure for quality does not exist in Norway yet. The closest I came to a course evaluation site was the commercial sites in the US, and specially the great success ratemyprofessors.com[14] (RMP) (see section 3.4). Their great success builds on the fact that students rate their professors on how easy it is to get a good grade. This is appealing to students, who use the site in big scale. A model like this is the opposite of what I want Kurskritikk to become. Where students on RMP write harsh and often immature critics, I want Kurskritikk's students to write good and fair evaluations. In order to achieve this, a working quality measure algorithm and some motivation for the students are needed.

When it comes to the quality measuring there is basically two ways of doing it. The first is user dependent and is built around users rating each other's reviews and the relationship between them. The second is independent of the users and is about measuring the quality of the text using its semantics, lexical and syntactic features. I will go through these by analyzing one high quality article thoroughly on each topic.

At last, I will go through an article talking about experiences on today's review rating systems and see if there is anything I can pick up to make this rating system better.

2.4.1 The first angle - involving the users

In the article "Open rating systems" from 2004 [6], Guha present a model to identify high quality content from an open rating system. He goes through the theory of rating and ranking and describes a case study of Epinions.

The ranking algorithms have been used successfully in the world's biggest review site Epinions. Guha has since the late 80s had relevant positions [5] in companies like Apple,

[14] http://www.ratemyprofessors.com

IBM and Google. He has also co-founded Epinions and some other projects, so I believe he has the technical knowledge to propose a good rating system.

Definitions and challenges

Guha describes an open rating system as a site where anyone can publish both the content and the ratings on this content. The systems for publishing the content and ratings can be different though. On the other side, a closed rating system is where a group of pre-qualified editors publish the content. Yahoo! is an example of such a system, where they have a large directory of sites published by these editors. So is the Open Directory Project[15], which by its name can seem open, but in fact is pre-qualified by their editors.

Such open rating systems deals with large and fast growing amount of content. When users rate this content two problems need to be solved:

Aggregation: We need a mechanism for aggregating the ratings of many sources into a single ranking.

Meta-Ranking: With an open system there will most likely be variation in the quality of the ratings. Therefore we will need some sort of ranking and filtering performed on the ratings themselves.

The real world phenomena "Word of Mouth" and "Web of Trust" is often drawn into such rating systems. People hold beliefs and make decisions based on their relations with other people, what we call trust. In Guha's model he describes trust as one of the central concepts. The ultimate goal is to make users trust the system, and where the system can generate this trust in the background while maintaining the anonymity of the users.

The Model

First he starts by defining a set of elements that build the model:

1. a set of objects O: $\{ O_1, O_2, O_3, \ldots \}$. These correspond to the objects being rated.
2. a set of agents A: $\{ A_1, A_2, A_3, \ldots \}$. These are either authors of content or raters who rate the content.
3. a set of possible values for ratings of objects D: $\{ D_1, D_2, \ldots \}$
4. a set of possible values for ratings of agents T: $\{ T_1, T_2, \ldots \}$.
5. a partial function $R : AxO \longrightarrow D$. This corresponds to ratings given by various agents to various objects.

[15] www.dmoz.org

6. a partial function $W : AxA \longrightarrow D$. This corresponds to ratings given by various agents to various agents.

In D and T Guha proposes either a restriction to $D = \{Positive\}$ or $D = \{Negative, Positive\}$. The latter set will impose some extra challenges as we will see later. A labelled graph W, the Web of Trust, can either be obtained by each user explicitly making statements of trust (or distrust) or by a system gathering this information automatically based on e.g. mining bibliographic databases (for citations and co-authorship). In Kurskritikk this can be done by interpreting a positive rating of a review as trust towards the user who wrote the review.

In the context of the model Guha set, he maps the identification of high quality content into two distinct problems that his model can solve: Rating and Ranking.

Rating: We want to complete R using W, i.e., assuming that Jane has not rated O_j, predict what her rating would be, if she were to rate it.

Ranking: Often we do not have one explicit object we want to rate, but a set of objects. This is where the ranking comes into action. We define a number N and pick the top-N objects and present them in order.

Calculation algorithms

In my case the ranking problem is the most important one, and Guha presents an approximate algorithm which has proven to work quite well in the context of Epinions:

> Given a user A_u who trusts $\{A_{ut1}, A_{ut2}, \dots\}$, distrusts $\{A_{ud1}, A_{ud2}, \dots\}$ and a set of objects O_s $\{O_{s1}, O_{s2}, \dots\}$, where at least some of $\{A_{ut1}, A_{ut2}, \dots, A_{ud1}, A_{ud2}, \dots\}$ have stated ratings on the objects, we need to compute the top N rated objects in O_s. Under the belief interpretation assigned to trust statements, we have to select the N objects with the highest probabilities of having Positive ratings.
>
> We do an iterative deepening (up to a pre-specified number of levels, typically 3) traversal of the Web of Trust graph. We first traverse all the agents directly trusted by A_u, then the agents they trust, who are not distrusted by A_u, and so on. At each level of the iteration we collect all the objects (in the set O_s) which have been rated along with their ratings and aggregate the ratings into a cumulative score for each object. Every Positive rating by a trusted agent adds a point to the score, every Negative rating by a trusted agent deletes a point from the score. In each pass we pick the objects with a score more than a preset threshold of scores. Typically, this threshold is just 1, i.e., we just need one of the agents in Jane's Web of Trust to certify that a piece of content is good. We stop when we have N objects.

While this works great for users who have a rich Web of Trust, it will not work as well for the rest. In most real world systems a substantial number of users will be anonymous

to the system. They are either new to the system or have for some reason expressed few or none statements of trust. To handle this, Guha defines global trustworthiness.

He first computes something he calls TrustRank, which combines trust and distrust in one single measure.

$$TrustRank_{N+1}(A_u) = \sum_{v \in T_v} TrustRank_N(v)/N_v - \sum_{u \in T_u} TrustRank_N(u)/N_u$$

A problem encountered using this formula is that distrust becomes analogous to negation. If Alice distrusts Bob who distrusts Jane, Alice trusts Jane. Further, this approach does not distinguish between an agent with small amount of (dis)trust statements (somewhat unknown) and one with a lot of them (controversial). What I find more interesting and relevant is where he splits these up into TrustRank and DistrustRank:

$$TrustRank_{N+1}(A_u) = \sum_{v \in T_v} TrustRank_N(v)/N_v$$
$$DistrustRank(A_u) = \sum_{v \in B_v} TrustRank_N(v)/N_v$$

The above formula calculates the trust and distrust for an agent A_u. T_v is the set of agents who trusts A_u and B_v the set of agents who distrusts A_u. N_v is the normalization for the number of people (dis)trusted by v. With two ranks, our ranking possibilities become more flexible. We can look at agents with high TrustRanks by themselves, ignore those who also have high DistrustRank or combine the two.

If a user has some local trust data but not enough to pick out the top N objects, then we could combine it with the global trust. Say N = 10, and the local data is sufficient to pick out a top 5. The global trust data can then fill up the remaining 5.

Angle summary

Guha show a lot of insight in the area of content ranking. The fallback on global trustworthiness is very useful for all kind of sites, as it will almost always be agents without trust data and/or content without ratings.

The separation of trust and distrust opens for more flexible use of the statements, depending on the application using them. To take full use of algorithms like these, the application needs lots of trust data from the users. This can be indirect statements such as ratings on content from agents, or direct ones like a user maintained list of eg. favourite authors. In Kurskritikk I only have thumbs up/down made anonymously on evaluations. These evaluations have a link to the author in the database, and the thumbs can therefore be used to build an indirect WOT.

2.4.2 The second angle - analyzing the text

While Guhas article was on rating and ranking content based on meta-data provided by users, Kim et al. analyzes the contents semantic features to measure the quality/helpfulness in the article “Automatically Assessing Review Helpfulness" [14]. This

is important because some reviews will always have few or no votes. This can either be because of the reviews age or simply that few users have chosen to rate it.

The model

The task is the same as Guhas, rank reviews on their helpfulness, or quality that is basically the same concept. They start by defining a helpfulness function h:

$$h(r \in R) = \frac{rating_+(r)}{rating_+(r)+rating_-(r)}$$

where $rating_+$ is the number of people that find a review helpful, and $rating_-$ the number of people that find a review unhelpful.

Kim et al. is in this article trying to capture the helpfulness of a review by experimenting with different features in five classes: *Structural, Lexical, Syntactic, Semantic and Meta-data.*

Structural features are observations on the reviews structure and formatting. They looked at:

- length (`LEN`): total number of tokens
- sentential (`SEN`): the number of sentences, the average sentence length, the percentage of question sentences and the number of exclamation marks.
- HTML (`HTM`): the number of bold tags <b> and line breaks <br>.

The *lexical* features capture the importance of words observed in a review. They experimented both with unigram (`UGR`) (one word) and Bigram (`BGR`) (two words) and measured the $tf - idf$ statistic on each. Tf is the term frequency, the importance in one review and idf is the inverse term frequency, the general importance in all reviews.

They calculate it using the following formula:

$$tfidf = \frac{tfxlog(idf)}{N}$$

where N is the number of tokens in the review.

The *syntactic* features captures the linguistic properties of a review. They measure the percentage of tokens which is open-class, which is divided into nouns, verbs, verbs conjugated in first person, adjectives and adverbs.

In the semantic features, Kim et al. argues that bigrams will not perform well because of their length and sparsity. Instead they hypothesized that good reviews often will contain references to the item being reviewed and sentiment words (i.e., words that express an opinion such as “great course”).

Meta-data is the data on the data, so they are independent of the reviews. In this case we are talking about a star-rating of the item being reviewed, similar to the {positive, negative} rating Guha presented in his article.

Experiments and results

Before starting the experiments they did an extensive pre-processing of the datasets gathered from Amazon. They started with 821 mp3 players with 33,016 reviews and 1,104 digital cameras with 26,189 reviews. They then removed duplicates, both products and reviews and reviews with less than 5 ratings. They ended up with these product/review pairs for mp3 players and digital cameras: 736/11,374 and 1066/14,467. To measure the correctness of the quality ranking they used a 10-fold cross-validation. 9 folds where used to train their SVM system while the last where used to rank the review using SVM prediction.

As regression model they used SVM_{light}[9]. To extract the features mentioned in the model, they used Minipar[11] dependency parser to generate the *length, sentential, unigram, bigram* and *syntax* feature set. To evaluate the quality of a ranking they adopted the Spearman correlation coefficient [15].

After trying various combinations of features with various results, the combination `LEN+UGR+STR` proved to be best with a score of 0.656 out of 1. The single feature scoring highest was `UGR` with the mp3 players and a score of 0.593. An interesting finding was the correlation between length and gold standard (star rating) in the Amazon reviews. Out of 5,247 reviews that contained more than 1,000 characters, the average gold standard for helpfulness was 82%. The 204 reviews with less than 100 characters had an average score of only 23%.

Their experiments also revealed that their features structural, sentential, HTML and syntax did not show any significant improvement in system performance.

Angle summary

This is a very important article as it will be useful to almost every rating system, with few or many users. There have been done studies on other aspects of helpfulness of reviews, such as predicting ratings [12] and measuring sentiment polarity and subjectivity [19]. The helpfulness aspect Kim et al. is studying by analyzing the content has there been published little or no research about.

They have collected a big and comprehensive sample of reviews to test on. They chose mp3 players and digital cameras, two very technical categories. I would have liked to see the algorithms tested on a softer category such as professional and technical books, where the reviewers probably will state some objective experience on e.g. the teaching value.

They do claim that their findings with length, unigrams and stars provide a basis of comparison for assessing helpfulness of reviews in other entity types. I do agree at that point since these are very general types of measures and therefore should work. A comprehensive and useful review can e.g. not contain only 10 words.

Chapter 3

Existing systems

We have now covered the theory of today's trust and reputation systems. To get a better understanding of how it actually works, let us take a look at some real world examples.

3.1 Ebay

The world's biggest auction site [1] with probably one of the best known and easiest trust and reputation systems. Ebay uses a centralized, person/agent and global reputation system. Here is how it works. After a finished auction, the two involved users can give each other one rating, -1, 0 or +1. They can only do this when they have done a deal, and only once per auction. All ratings of a user add up to a rating number, where higher positive is better. This rating is visible for all members, so are some statistics, such as how many ratings is positive, and ratings for the last month, 6 month or year. You can also see ratings on the user as a buyer, seller or ratings s/he has given.

This system has some flaws which decreases the usefulness of the rating system. People are reluctant to give negative ratings as the other user can see your rating and therefore be afraid of revenge[2] Only 1% of the ratings on Ebay is negative and less than 0.5% neutral [17, page 9-10]. Another problem is that users with a bad total score easily can change identities by registration as new user under a new alias. Ebay has a lot of data they could have used to make the system better. One of them is data attached to the ratings such as the value of the product and total ratings as a seller and buyer. Another important element is the reputation of the raters. If you get 10 ratings where all raters have a negative total score, then those ratings probably should not have the same influence as ratings from highly rated users.

To cope with the fear of negative comments Ebay could implement a system with a preset feedback time. The ratings would not be published until this time had elapsed.

[1] http://www.ebay.com

[2] As of May 2008 Ebay will change their rating model. The sellers will no longer be able to give negative feedback to buyers.

In this way users could give comments without the fear of revenge. Ebay does probably not do this as more negative ratings decreases number of deals and hence decreases Ebay's income.

3.2 Epinions

Epinions is a web site focusing only on rating and reviews of various items such as books, movies, electronics, etc. It is classified equally as Ebay, except Epinion is not a person/agent, but a resource reputation system. The users are providing all the content which is reviews on items as well as ratings on the reviews themselves.

The items are organized into categories. In each category, the users are divided into 5 groups; *category leaders, top reviewers, advisors, most popular reviewers and ordinary members*. The category leader is in charge of the category and also chooses the top reviewers and advisors. Top reviewers are members who write reviews of high quality. Advisors help the shoppers find the right products by rating reviews and give constructive feedback to users on how to improve the quality of their reviews. Ordinary members are the default starting state of a user.

The goal of Epinions is to help users find good products, and in order to do so they need a lot of help from the users. The motivations given to the users are; building a user friendly site and their effort is made visible through the member hierarchy. They also pay reviewers based on how often their reviews are read and if they are used in a buying decision and other campaigns. In January 2008 they had a $10 for 10 reviews campaign where the requirement was at least 200 words and relevancy.

All these factors result in ordered lists of reviews, placing those from the best reviewers on top to ensure that the user get the best reviews for the product. They depend on human effort to maintain the system, and it seems to work well for Epinions.

3.3 LinkedIn

In social web sites the trust is not only based between two persons. LinkedIn [3] is one example where they build on a model where you trust a person through one of your connections. Let's say Alice is connected directly to Bob, and Bob to Charlie, then Alice can choose to trust Charlie through Bob. In addition to just connecting to people, you can also write recommendations on people you know, making them more valuable or trustworthy to people not connected directly to them.

LinkedIn offers several ways of using the web of trust to benefit you. As an employee you can search jobs or maybe find co-workers you want to start a new business with. As an employer you can list available jobs and search for competent people. These people can be connected to you directly and in second or third degree, which can make them more trustworthy. Employees can choose to open up their profile for employers

[3] http://www.linkedin.com

so that no connection is needed to start a dialogue. You can list people on how many recommendations they have received, and filter this to recommendations from all users or limited to your network. This is called 'Services' and is intended to be used to find people who can provide a given service with good recommendations backing them up. A feature which benefits all users is 'Answers'. In here all users of LinkedIn can post and answer questions. The clear advantage here compared to eg. a discussion forum is the fact that we can evaluate the educational background of the users, thus appreciate a more correct answer to the question.

LinkedIn is a professional tool and is based on full identities of the users. They use these identities and relationships in between them to build many useful features. As a business/employer you can pay a premium and get easier access to more people. This works well because LinkedIn has grown to become a very important network with approximately 19 million users (March 2007).

3.4 RateMyProfessor

RateMyProfessor.com (RMP) is a huge rating site, and its function is what the URL indicates; rating professors. Per May 2008 the site has 6 million ratings divided on 1 million professors from 6000 different schools, all this divided by the United States, Canada, England, Scotland and Wales. Their aim is to provide the students with course information simplifying the student's decision making, the same goal as Kurskritikk strive to achieve.

In 2006 the two university employees Davison and Price did an evaluation of RMP[3]. After analyzing the results of their study, I was surprised to see how low the quality of the reviews was. Instead of evaluating the quality and outcome of the learning experience, the students concentrate their reviews on easiness of courses and sexual appeal of instructors. RMP is a commercial site and does little to hide this fact. The site is designed to lead the reviews in such a direction. In RPM the students can evaluate the following; easiness, helpfulness, clarity, sexual appeal, and the student's overall interest in the class prior to taking the class. An overall score is averaged from the helpfulness and clarity ratings. High overall scores produce the symbol of a smiling face next to an instructor's name; low overall scores produce a frowning face. Similarly, instructors with high hotness scores receive a chilli pepper next to their name.

The findings from the study show that easiness of a course correlates with the overall score. The general understanding of RMPs users is that they behave as consumers shopping for courses and degrees. This is far from the reality in Norway and UiO where the focus is on the quality of lectures, curriculum, lecturers, tutors/instructors and the implementation of a course.

There are other sites in the US, but with far less users. Some of these are pickaprof.com, campusdirt.com, myprofessorsucks.com, and rateyourprof.com. The names of some of these indicate a clear non-academic focus. The fact is that in the US they do not have the same use of external examiners as in Norway. The students therefore put more effort in the professors and instructors, as they have more influence on the grades.

Chapter 4

Kurskritikk - the web application

In the appendixes you will find a full requirements specification and system documentation. It is not easy to get a good and quick overview by reading those, so I will now shorten it down and describe the system from a higher level.

The site can be tested with all its functionality with these credentials:

- **URL:** http://www.kurskritikk.no/
- **Username:** testuser
- **Password:** usertest

4.1 Purpose and overview of the site

I have already mentioned how course evaluations are given at UiO today and some about how I feel Kurskritikk can apply something new into the process. Now I want to discuss the purpose of the site in the context of functionality.

Kurskritikk is first and foremost a contribution to the students. It is the students who give the evaluations and I feel they should get something back. In Kurskritikk students can give evaluations with ratings, and they can mark an evaluation written by others as helpful or not helpful. They can only do this once for each evaluation, which is controlled by sessions. I chose this over IP-logging and registration to give everyone the possibility to rate the evaluations. Figure 4.1 shows how the front page of Kurskritikk looks like today.

At registration the users will have to provide the following information:

- Username (their UiO-username)
- E-mail address
- Password

Figure 4.1: Kurskritikk.no - front page

The users will have to register with their UiO-username in order for Kurskritikk to verify that they indeed are UiO students. After registration an email is sent to [username]@ulrik.uio.no which will forward the email to [username]@student.[faculty].uio.no. The users are anonymous in the front-end. Each user will be able to manage a profile (see section 4.3 on the facing page).

4.2 Motivation

In order to get users to write evaluations, they need some kind of motivation. Epinions has an income through their advertisements and can have a model where they pay their top contributors. Kurskritikk is an open site with no income and is based on voluntary contributions from students. Kurskritikk will help students choose courses and will help administrations improve the courses. If you write serious and good evaluations which many users find useful, it may give the author a feeling of helpfulness and achievement.

If many users contribute to Kurskritikk, it will become a better service for students and administrations, and I think most students are aware of this. It is different from community to community, so it is not easy to predict before it is tested. I am positive since today's students have a close relationship to social websites and sharing of everything from meanings on products, pictures to their everyday life. By taking the evaluations a step closer to a social website I think more of today's students will get a more active relationship to the whole concept of course evaluations.

4.3 Functionality

The functionality of Kurskritikk will be as easy as possible while still offering the needed services to help students find information about courses. In Kurskritikk, users can do the following without logging in:

- **Browse courses** by name, course code, faculty and institute
- **Search for a course** (exact match on one course code will forward the user directly to the course page). The user can search on course code and name.
- **Show evaluations** for a course and sort them by course code, name, credits or a built in quality measure algorithm.
- **Give a thumb up or down** on an evaluation (useful or not useful)
- **Create an account**
- **Reset password** (by providing the email address of the user)
- **Log in**

By logging in, the users can also write evaluations and manage a profile. The profile will have the following functionality:

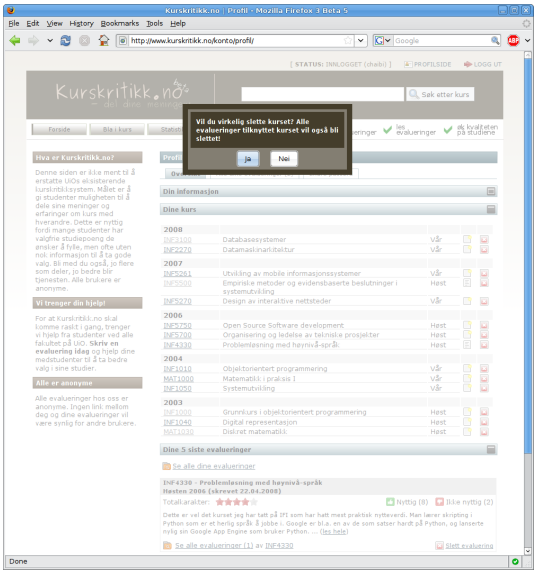

Figure 4.2: Kurskritikk.no - Profile

- **Overview of personal data**: username, faculty/institute (can be modified) and e-mail (can be modified)
- **List of courses they have taken**, with a mark if they have written an evaluation on it (can be deleted)
- **Evaluations they have written** (can be deleted)

- **Password change**

I have chosen to let the user control their submitted data. A user can delete every evaluation s/he writes. The evaluations can be deleted from all its views (eg. from the profile or from the front page). This is how Facebook handles user contributed data today, and it seems to work well. On Facebook the users can delete all wall posts, images, image tags and so on. This gives the user a better feeling of control and may help to bring the threshold of writing an evaluation down.

4.3.1 The evaluation

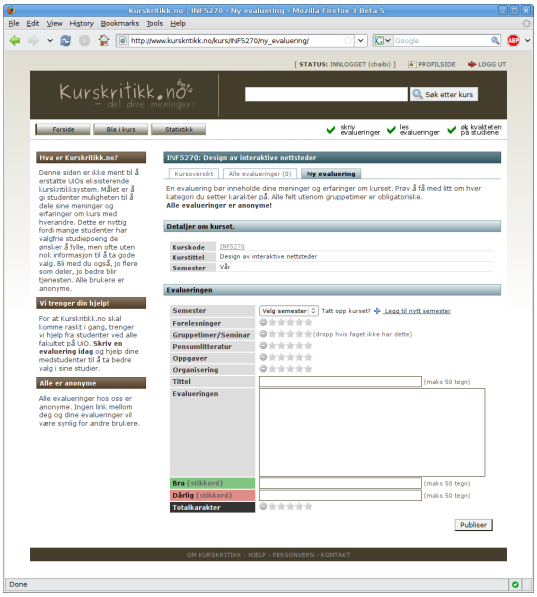

Figure 4.3: Kurskritikk.no - New evaluation

The design of the evaluation form is inspired from several big sites that provide similar functionality. These were among others Epinions, Amazon and Hardware.no. I primarily used Epinions as a reference, together with a set of requirements I want to fulfil:

- Make it easy to write an evaluation
- Get a rating on all key elements of a course as well as an overall rating
- Attach the evaluation to a semester
- Take input that makes the evaluation easy to read for other users

This resulted in the following evaluation form:

- Semester

- Ratings: lectures, tutorials/seminars, curriculum, assignments and course organization
- A headline for the evaluation
- A single evaluation text
- Key words for positive and negative sides of the course.
- Overall rating

The overall rating is not calculated as the average of the 5 areas above because a user may feel that a course deserves full total score despite lower individual score. I keep the evaluation text to a single field to lower the threshold for contributions. I hope the rating areas will help the user think of the various sides of a course when writing an evaluation.

This is a well balanced set of elements I think will catch all elements and sides of a course and also keep the contribution level high. The alternative could be to force users to write a text for each rating area which would make the evaluation more clear, but at the same time make the participation threshold too high and lower the amount of evaluations. As it is now the user is freer to include/exclude elements as s/he wishes.

4.4 Under the hood

I have now scratched the surface of Kurskritikk's purpose and functions. To get a better understanding of how these functions actually work, I will now present some of the underlying concepts in more depth.

4.4.1 Django - the web framework

To keep focus on the thesis theory and Kurskritikk's functionality it is very helpful to have a high quality web framework to rely on. Django[1] is exactly this and is based on the Python programming language. Django defines itself like this:

> *Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design.*
>
> *Developed and used over two years by a fast-moving online-news operation, Django was designed to handle two challenges: the intensive deadlines of a newsroom and the stringent requirements of the experienced Web developers who wrote it. It lets you build high-performing, elegant Web applications quickly.*
>
> *Django focuses on automating as much as possible and adhering to the DRY (don't repeat yourself) principle.*

[1] http://www.djangoproject.com

This is pretty much what Django does as well. I had never used the framework before this thesis, and it did not take long before I understood how it worked and could take full advantage of its functionality.

Django is built using the Model View Control (MVC) principle. You define your database as model classes in a file called `models.py` (the model), the presentation is handled by HTML templates written in Djangos own template language (the view). To glue it all together we have the views which is stored in the file `views.py`. To control which view is called when, we have the last brick in the puzzle, the URL config `urls.py`. In this file all URLs of the web site is listed using regular expressions. If a URL match, an appurtenant view is called.

To get a better understanding of how Django works, I will now explain what happens when a user accesses a course page of the application. The user types in the URL `http://www.kurskritikk.no/kurs/INF1000/`. Django reads the HTTP request and makes a lookup in the URL config and finds the match

```
(r'^kurs/(?P<course_code>[\w\-]{3,15})/$', 'view_course')
```

At the end of the URL config line we will find the view which is called when the URL matches. The view `view_course` is then called to do the business logic. Inside the URL regex we have a course code in a variable (`?P<course_code>`). This variable is passed to the method a long with the HTTP request object. `view_course` then does its magic, and then passes the relevant data to the template. If the course does not exist, a HTTP404 error exception will be raised and then display a standard HTTP404 template with the sites design. The template gets its data in a Python dictionary which is like an array with lookup keys. The dictionary can contain single variables, new dictionaries or simple lists. Object orientation is active in all parts of Django, which means we can send the template a course object which the template can obtain the course name from: `{{ course.name }}`.

This only gives an overview of how Django works. Additional features includes

- Simple administration interface for the models.
- Form processing in a really simple way.
- User registration and sessions.
- Middleware extendibility (eg. implementing a blog app in middleware is simple)
- Caching, security and internationalization possibilities.

More on how Django is used in Kurskritikk in Appendix B on page 55.

4.4.2 Quality measuring evaluations

Based on the literature I have found on the field, I will look at alternative ways of measuring the quality on evaluation with a unique algorithm that can solve the ranking

issues. What's nice with both the technical articles ([6], [14]) is their consideration for general usage of their algorithms.

In Kurskritikk all users will be anonymous, so active building of "Web of Trust" will not be possible. I will let users rate the courses on a scale from one to five. I will also let the users rate the evaluations with a thumb up or thumb down (positive or negative). That combined with the length of each evaluation will be the basis of the quality measurement algorithm I am building for Kurskritikk. I could build an indirect WOT for logged in users which could be used as a variable in the algorithm. This is excluded due to time limitations of the thesis.

The algorithm explained

What I essentially want to do is to calculate a score between 0 and 1, based on a set of variables included in the algorithm. My variables are thumbs and length. Each of those will be calculated individually and also be given a score between 0 and 1. Default value is 0.5 for each variable, where the most negative outcome can drag it down to 0 and the most positive up to 1. A weight on each variable is multiplied with its score where the sum of all weights is 1. The weights are listed in table 4.1.

Attribute	**Weight**
Total	
Thumbs score total	0.65
Length score total	0.35
Total	1
Thumbs	
ThumbsQuantityScoreTotal	0.25
ThumbsPercentScoreTotal	0.75
Total	1

Table 4.1: Rating weights

Constants	**Value**
Thumbs down max (TDMAX)	The highets TD value of any evaluation
Thumbs up max (TUMAX)	The highest TU value of any evaluation
Negative length border (NLB)	250
Positive length border (PLB)	700
Max length (ML)	1700

Table 4.2: Rating contants

The thumbs score I divided into two parts. The first is based on the difference between positive and negative ratings. Default is 0.5 (when 0 thumbs is given or thumbsUp thumbsDown), and 100% thumbsUp adds 0.5 and gives 1, and 100% thumbsDown withdraws 0.5 and gives 0. The second is based on the quantity of thumbs. This is to differentiate between 1/0 and 10/0 thumbs up/down and between 1/10 and 10/100 where 10/100 should be given some more weight. The percent scores calculation follows:

$$thumbsPercentBase = 0.5$$

$$thumbsPercentCorr = \frac{thumbsUp - thumbsDown}{thumbsTotal/0.5}$$

$$thumbsPercentTotal = (thumbsPercentBase + thumbsPercentCorr) * weight$$

It is not enough to know how many percent thumbs are up and down. We need to take the amount of thumbs into consideration as well in order to separate some cases. A scenario is when two evaluations has the exact same length and evaluation 1 has 2/0 thumbs up/down and evaluation 2 has 8/0 thumbs up/down. The percentage of thumbs up will in both cases be 100, so the quantity is used to differentiate. This is calculated like this:

$$thumbsQuantityBase = 0.5$$

$$thumbsQuantityCorr = \frac{thumbsTotal}{(TDMAX + TUMAX)} * (thumbsPercentCorr * 2)$$

$$thumbsQuantityTotal = (thumbsQuantityBase + thumbsQuantityCorr) * weight$$

The total score for thumbs is then calculated as this:

$$thumbsPercentTotal = thumbsQuantityTotal + thumbsScoreTotal$$

As we can see both the percent and quantity score has a base value of 0.5. This will then increase up to 1 with a majority of thumbs up, and vice versa for thumbs down. The *thumbsQuantityCorr* is a bit special as it uses *thumbsPercentCorr* to tip its value in positive or negative direction. Two evaluations with the thumbs up/down 20/2 and 2/20 should have the same absolute value of textitthumbsQuantityCorr added or subtracted from the base value of 0.5.

The same model with a baseline of 0.5 is used with the length as well. All evaluations with length between NLB and PLB gets the score 0.5. If a length is under NLB it is withdrawn up to 0.5 points, depending on how near 0 characters it is, including spaces. If the length is over 700, up to 0.5 points is added to the score, depending on how near ML it is. Both graphs are linear. This is how the length score is calculated:

If length is under NLB:

$$lengthScoreTotal = \frac{length - NLB}{NLB} * 0.5$$

If length is over PLB:

$$lengthScoreTotal = \frac{length - NLB}{ML} * 0.5$$

As with some of the fractions above where the numerator can exceed the denominator, the score is freezed at 1 here as well if that happens.

The constants, weights and max/min case

So how does an evaluation achieve a score of 1 or 0? These scores are a result of the values in table 4.4. Since an evaluation has to contain at least 1 character, the total score will never reach 0, but will come pretty close.

Attribute	**Min**	**Max**
Thumbs down	TDMAX	0
Thumbs up	0	TUMAX
Length	1	$\geq ML$
Gives the rating	$\sim$ **0**	**1**

Table 4.4: Rating Min/Max

The literature I have seen does not say much about the implementation of their algorithms, but concentrate on the concepts and variables. Based on this literature and my application I found user ratings (thumbs) and text length to be the basis of my algorithm. As I am only looking at one variable of the text, the length, I chose to not let this get the highest weight. As it is very easy for users to give a thumb up or down on an evaluation, and only can do this once[2], this should have the biggest weight.

Why these constants (table 4.2)? Let us start with the length. In order for an evaluation to cover enough topics to be useful it needs to be over some length. The magic number for this length is not easy to define, so I had to analyze the text of some evaluations to get an idea. What I experienced was that the evaluations with less than 250 characters started to lose value. An evaluation should cover as many aspects of a course as possible to be of good value to the readers. These aspects can be those I let users rate a course on; lectures, tutorials/seminars, curriculum, assignments and course organization. With less than 250 characters that's just not possible. So for every character under 250, the length score gets closer to 0.

Between 250 and 700 characters I have defined a free space where all evaluations gets the default score 0.5. This is because it is hard to differentiate between these lengths. An evaluation of length 350 can be as good as an evaluation of length 600, it depends on the writer. But when an evaluation reaches a length of 700, it most likely means the quality and coverage of aspects is better. So from 700-1700, the evaluation can increase

[2] As long the session data in the browser is not deleted.

the length score from 0.5 to 1, with a linear graph. When the length reaches 1700, there should be enough characters be a max useful evaluation.

For thumbs I have chosen variable constants which always will correspond to the evaluation with the highest value of thumbs up or thumbs down. When it is done like this the algorithm will be able to differentiate between evaluations with thumbs of any value.

The weights are set to highlight the importance of each variable in the total score. Humans are still the best to evaluate text, and therefore the thumbs score get the highest weight of 0.65. But thumbs are not enough by itself. We may have situations where evaluations have equal partition of thumbs or where some thumbs are a result of some sort of rating mistake from a user. With a weight of 35%, the length can correct such cases.

Inside the thumbs score the weights are divided into two parts, one for the percentage of thumbs up or down, and one for the quantity of thumbs. The percentage has the largest weight (0.75) as this should be the basis of the score. An evaluation with 75% thumbs up should always win over one with only 50% thumbs up. But when the percentage is equal, the quantity of thumbs makes the little difference. With a weight of 0.25, it will make it possible to differentiate in such cases.

Chapter 5

Results and discussion

With theory and description of Kurskritikk covered, it's time to take a look at what achievements have been made. I will also discuss pros, cons and potential expansions and/or changes that can be made.

5.1 Sorting evaluations

To make it possible to develop a good sorting algorithm, I needed a lot of test data. One alternative was to generate this data by some sort of script, or to get a group of users to write a set of evaluations and rate them as well. I wanted a realistic data set, so I chose the last option. As I started the development, I realized that this could end up becoming a fairly good site. I raised my goals and decided to build a fully functional site which could run after the thesis was completed.

When the site was built and released in the domain kurskritikk.no, I started by recruiting fellow students to write evaluations. I gathered around 20 of them. Then I contacted all the profession committees at UiO and the UiO student newspaper. They wrote a story about the site, and suddenly I had around 70 evaluations and around 200 ratings (thumbs). This gave me enough data to do some good tests on my algorithm.

5.1.1 Results

To get an overview of the results, I grouped the evaluations by intervals of its length. I have short intervals at the lower lengths to make the low rated (the length score) more visible, while the bigger lengths have larger intervals. On each interval you can see the average number of thumbs up and down, and you can see the average total score. The results can be viewed in figure 5.1 on page 37.

As we can see on in figure 5.1 on page 37 there is a relation between length and total score. As the length increases, the average score for the intervals does so as well. The user's ratings correspond with the length of the evaluations. These results also

Sums	
Variable	**Count**
Evaluations #	71
Sum Thumbs Up	190
Sum Thumbs Down	40

Total averages	
Variable	**Average**
Average score	0.69
Average thumbs down	0.56
Average thumbs up	2.68
Average length	549,89

Table 5.2: Total sums and averages

correspond with the results produced in the study by Kim et al. Users seem to be a bit careful with the thumbs down (see 5.2, with an average of 0.59 out of 1 thumbs down per evaluation, it may seem that they just do not have any negative feelings about the evaluations. However, as the sample size is very small one cannot draw any conclusion on why this is. As a result of low use of thumbs down, the average score might be a bit higher than expected, at 0.68. This will most likely adjust as more users take advantage of the thumb function. Anyway, the high average score might also be a result of overall high quality on the evaluations. The site needs a bigger sample size to generate more useful and realistic statistics.

All evaluations sorted on quality can be viewed in table C.2 on page 69.

5.1.2 Future improvement possibilities

Using a Web of Trust

In Kurskritikk the users cannot build an active web of trust. However, building one indirectly where the WOT is filled with users based on the thumbs they give could be a feasible solution. If one gives a user a thumb up, we add the user to the WOT. How much we trust a user could depend on how many percent of the thumbs we give the user is positive. This can also be expanded to a negative scale where we distrust users who write evaluations we give thumbs down. This can only be achieved for logged in users as we need to save the WOT data in a database with links to the regarding users.

I can now use the calculation algorithm Guha presented and build a score for each evaluation. Three steps down the WOT as he suggested will be a reasonable level of depth. This is what gives or takes points from a course in such a ranking:

- a user in U_1ss WOT has written a review, give it points

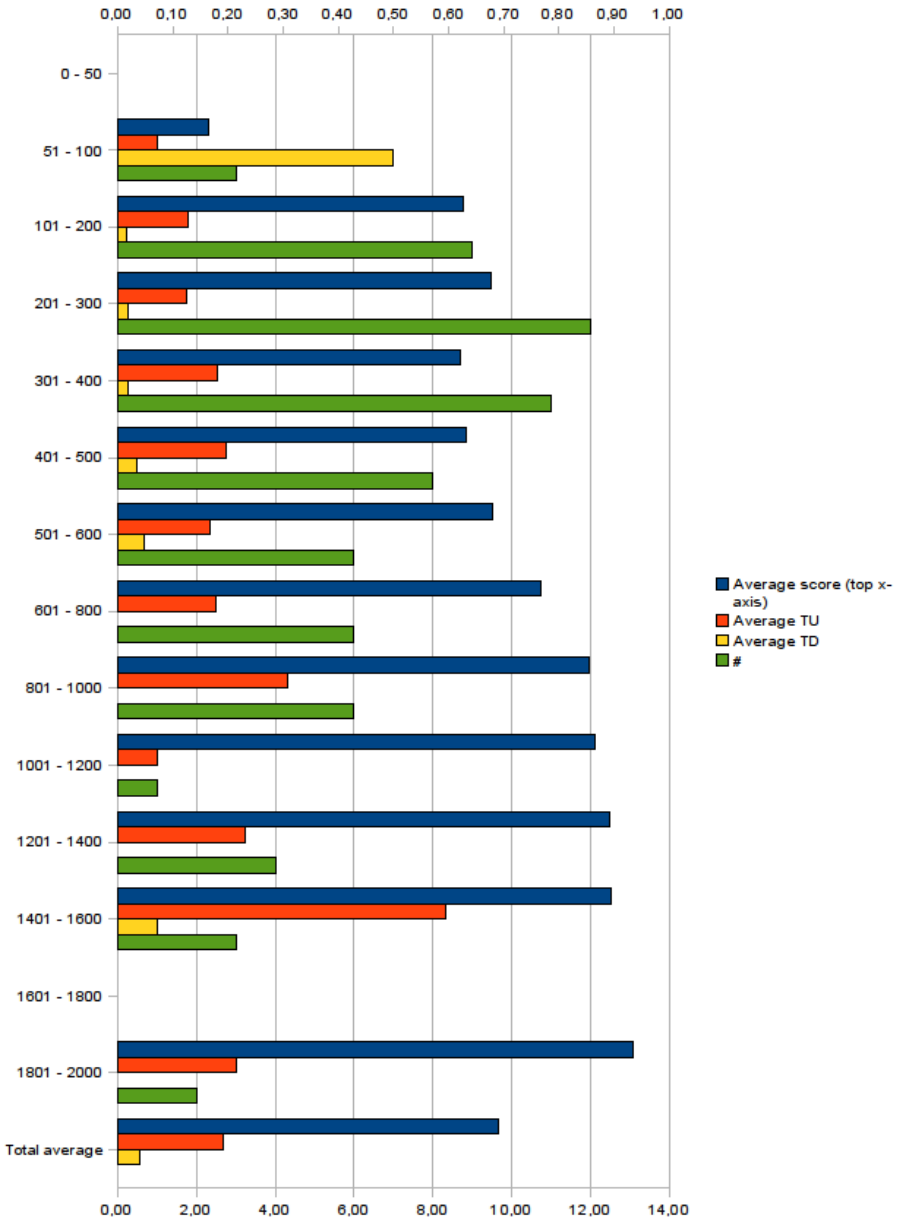

Figure 5.1: Statistics (20.05.2008)

- a user in U_1s WOT has rated a review positively, give it points
- a user in U_1s WOT has rated a review negatively, withdraw points

The further down we go in U_1s WOT, the less weight it should be given. A global trust score can be used to fill a top N list of evaluations if the local WOT does not suffice. We can then apply the same algorithm on the users outside the WOT who have evaluated or reviewed an evaluation on the course U_1 is looking at. The global trust can kick in if there is not enough or no trust data on an evaluation directly connected to the user requesting a ranking. I think this is necessary since a user's lifetime in this system probably will not be longer then her/his education.

The WOT score should of course be combined with the thumbs and length as they are implemented now. The effect of this indirect WOT will only be speculation. Just because one user gets a thumb up on one evaluation does not mean that s/he always writes good evaluations. A good WOT is dependent on a large set of users and thumbs.

More elements of the text

In the existing algorithm explained above the only element of the text used is its length. There are a couple of other elements we could take in as variables as well. Kim et al. mentions a couple of extra text features that can be analyzed:

- Number of sentences
- Average sentence length
- The percentage of question sentences
- Number of exclamation marks

In addition to this, my experience with online forums, rating sites etc, is that the quality can suffer from the style of writing. This can be measured by looking at elements such as:

- All the elements above
- Space after ended sentence
- Capital letter at the start of every sentence

The message of the evaluations could still be good and of high quality, but the readers often put a lot of weight on the wrapping. Reading text with many errors in the grammar and a bad style of writing ruins the reading experience.

5.2 Implications on use of identities

In the theory section I described different ways of ensuring users privacy. In Kurskritikk I use a model where the users have total anonymity in the front-end, but no anonymity in the back-end. In fact, the username they have to provide in order to authenticate

as UiO students can be used to get their full name. This model was chosen because I wanted to authenticate the users as real UiO students without having to use any connection to UiO's systems, but still give the users full anonymity in the front-end. Their real identities are hidden. It can be done this way because the evaluations the users write are not highly sensitive data. It is of course interesting for some people to know who the authors are, but it will most likely be more curiosity then for hostile use.

After some rounds of thoughts on my model I see that the UiO username does not need to be stored in the database. It could be used only to send an activation mail, and never be stored any place. The users could then choose a new username/alias which they would use to log in to the site. With this model the users would still only be anonymous in the front-end, but the 100% identifiable UiO-username would now be replaced with an alias and an e-mail address.

We can place the models in two categories; linkable and unlinkable. When we say linkable, we talk about the back-end and connections between evaluations and users in the database.

5.2.1 Linkable

Linkable models can never offer full anonymity in all parts of the system. You can chose to hide the identity in the front-end, but there will still be a connection in the database. We differentiate between linkable models by looking at how the identities are handled in the front-end. There are three main paths.

Full identity will show a full name ala the way Facebook does. The dangers in this context are the possibilities of using evaluations against the authors if the professors reading them have any influence on grades or approvals of assignments. I would not believe that to be likely, but we should consider the possibility. A consequence of this would be users who may resist writing negative evaluations because of that danger. It would lead to fewer evaluations with a more gentle language most likely (ref. section 2.3.1 on page 12) and higher quality. It is more likely that the users are more matter-of-factly. Advantages of the model are possibility to let the users build an active Web of Trust and build more social features around the content they provide. It is often easier to trust the author when you see his/her full name.

Alias as discussed in the theory is how Ebay and Amazon implements identities. The user picks an alias that will be visible for other users. It has the same disadvantages and advantages as a full identity, but with less weight. The users are identifiable, but not directly through their alias. Other users can identify them if the alias contains parts of the name, or they can look at what courses they have taken and identify those who have unique or rare combinations (fingerprinting). Building of WOT is still possible, but the social possibilities are weakened.

Hidden identity is how Kurskritikk implements identities. Visible active WOT is not possible, but the other elemements such as evaluation seriousity and quantity is well balanced.

A model that combines these could let the users choose which level of identity they want. The various levels could then imply different features. Users who chose to write under full identities could be awarded with possibility to build a WOT. Their evaluations could get more points in the sorting algorithm, a model Slashdot[1] uses. As I have mentioned, authors who identifies themselves with full name will earn more trust among most people.

5.2.2 Unlinkable

Only unlinkable evaluations can give true anonymity. In these models the evaluations will always be unlinkable in the front-end. In contrast to linkable models, these will not have any link to the evaluations in the back-end either. There are two main paths:

With pseudonyms and a third party the evaluations will not be directly linkable to a user. As explained in the theory only the two systems combined can retrieve the author of an evaluation. I place this in the unlinkable category because the probability of a link to be known is very small as both systems would have to be compromised at the same time in order to achieve this. The advantages in this model are found in the back-end. With a persistent identity attached to all evaluations, an indirect WOT could be built to present more valuable and relevant information to the users. The disadvantage is that pseudonymity is complex and costly. It requires an independent third party and an advanced implementation. It is clearly not in the scope of this thesis, but would definitely be an option with more time and resources.

User generated pseudonyms As mentioned in the theory, a model could be letting the users registrate with a username they pick themselves. They could still provide their UiO username for authentication purposes, but after they are authenticated as UiO students, the only information stored on the user would be the username they picked. A clear disadvantage would be losing the possibility to send password reminders. This is a central feature in a web application that is seldom used. Kurskritikk would probably not be used more than a handful times per semester, and it is likely to assume they will forget their password at some stage.

Total anonymity is where the evaluations are stored completely independent of any other tables in the database. No kind of possible identifiable information is attached to the evaluations. The users can still log in, but it will have no effect on the anonymity matter. It would be the same if all users could write evaluations, without logging in, as it is with article discussions in some online newspapers.

[1] http://www.slashdot.org/

5.3 Taking advantage of the social platform

A web application such as Kurskritikk can take advantage of implementing its functionality into social platforms such as Facebook and Open Social. Many UiO students have Facebook profiles and are part of the network Uni.Oslo. Ratemyprofessors.com has made such an application for Facebook[2], and users can now add the app in their accounts. The application lets users search for professors and read the first ratings on them. If they want to see more ratings, they are sent to ratemyprofessors.com.

As writing evaluations is the most important task at Kurskritikk, it could also be implemented into a potential Facebook app. The user would then have to be authenticated through some kind of login, or adding a unique code in the applications options.

5.4 Possibilities with Kurskritikk

Kurskritikk has a database filled with data that can be used to generate a lot of interesting new data. Amazon has the function that tells users something like "users who bought this item also bought these items..". The same can be done with courses at Kurskritikk. It depends on enough users and added semesters in their course list, but with that we could generate data like that. Another feature could be to let users share their course lists (still anonymous) and let users reading an evaluation see the background of the evaluator. It is sometimes useful to see if the user evaluating a course has similar background as yourself.

The courses in the database is today collected through a manually executed script which collects attributes such as course name, code, semester, credits, faculty, institute and url. These data are limited to the data in the XML documents from UiO. Courses could get added data such as course abstract, background requirements and so on. This data must be collected by parsing the courses HTML site and would have to be updated every semester in order to guarantee fresh data for the users. By adding extra data and providing this through a good browse and search interface, the users get more functionality. This leads to more reasons to use the site which again leads to more users and hopefully more evaluations.

[2] http://apps.facebook.com/myratemyprofessors/index.jsp?id=1

Chapter 6

Concluding remarks

In this thesis I have successfully built the web application Kurskritikk which gives students at the University of Oslo possibility to write course evaluations open for everyone to read. In Kurskritikk I have implemented quality measuring algorithm which is used to order evaluations on quality. When the amount of evaluations gets big, it is important to be able to display the best and most useful ones first.

The algorithm is built around a set of variables with various weights. These variables are divided into two main categories, meta-data and text analysis. In Kurskritikk I use useful ratings (thumbs up/down) and the length of an evaluation. The algorithm has been tested on 72 evaluations and the results show reliability. There is a relation between the score and useful ratings, which show that the users agree with the algorithm. The sample of useful ratings is too small though to draw any conclusions.

Different types of identity management models have been discussed, and a hidden identity model was implemented in Kurskritikk. In this model the user's identity (a username and e-mail) is linked to the evaluations in the back-end, but hidden in the front-end.

Kurskritikk is developed using the Python web framework Django. As a developer with almost no experience on this kind of web development, I still managed to implement a fairly good and functional application. Using a framework like Django gives the developer opportunity to concentrate on the design and functionality, and not on all the back-end details such as databases, sessions and performance issues. My experiences are very positive.

During the work on this thesis I have built great knowledge on implementation of quality measure algorithms. The support of a solid framework such as Django has proven to be invaluable. Even with the support from Django I had to exclude some initally made feature wishes, where the most significant one was implementation of a Web of Trust. The work has increased my interest in the field and motivated me to continue the work on the application.

Students from the University of Oslo now have an open alternative for course evaluations, and hopefully more students will take advantage of the service and make it more

useful to all students.

6.1 Future work

It is difficult to predict how the algorithm developed will work when the evaluation count increases. There is however improvements that can be made to improve the general performance. These include more thorough text analysis and the building and usage of a Web of Trust. As a web application there will always be small and big improvements that can be made. One example is more aggregation of data, for example displaying how many users each course has or building statistics such as "users who have completed course X, also have completed course Y". When Kurskritikk has settled at UiO, an obvious thought would be to extend it to more universities in Norway. The goal should be to improve the quality of studies for as many students as possible.

Appendix A

System requirements specification

This section will cover all the requirements I have set for this site. This includes functional as well as non-functional requirements. I will use the OPEN Process Frameworks (OPF) template for system requirements specification (SRS). The OPEN Process Framework (OPF) is a practical, public-domain, industry-standard, general purpose management and engineering process framework that is primarily intended for the object-oriented, component-based development of software-intensive systems. OPF is developed by the OPEN Consortium which is a non-profit group of over 35 dedicated individuals spread around the world, whose membership includes international-recognized methodologists, consultants, academic researchers, CASE tool vendors and users.

Since Kurskritikk is developed by me only, I will not write a full SRE, but extract the parts needed to get an overview of the system and how it is supposed to work. Kurskritikk is written using Django.

A.1 Introduction

Kurskritikk is a medium advanced web application with a lot of small parts building the whole. This SRE will give an overview of the system and explain the critical and important parts in some more detail. The reader needs a basis of knowledge on web applications to take full advantage of the SRE. I will start with an overview and extract it down to parts.

A.2 Goals

Kurskritikk has a goal of bringing the students opinions and experiences out to every student. Students will be able to register and write evaluations. The users will own all the content they produce, and be able to delete it. The evaluation can be rated as useful and not by all visitors of the site. These ratings are then used as a part of a sorting algorithm used to order the evaluations on quality.

Kurskritikk should be up to date on all courses and have a appealing and easy to navigate user interface that stimulates to further use.

A.3 Functional requirements

A.3.1 front-end

All users will be able to register an account, but most of the information will be available without logging in. As Kurskritikk has a simple and easy to navigate structure and design, I will not explain all minor details. I will begin with the functionality all users will have, independent of login status and then go through the front-end functions for logged in users. These are described using use cases.

Overview

In Kurskritikk the evaluations and courses are in focus. Both evaluations and courses should be accessible from several places in the site. Course codes should be links to the courses overview site, and functionality such as rating and deleting evaluations should be active in all pages where an evaluation is displayed. All this follows the new trend of web applications where the user has interesting and relevant links and functions from many locations within the site. This should be combined with dynamic content, especially on the front page, but a sidebar can also be used to accomplish this in sub pages. The functionality of logged in and not logged in users will now be presented.

Not logged in

Not logged in users will be able to search courses, view and rate evaluations. For a course the user can list all evaluations, and sort them on various variables. In order to register an account, the user's needs an active username from UiO, which gives the functionality of logged in users (see A.3.1 on page 48). The following tables describe the most important use cases for users who have not logged in.

Search courses	
Goal	Find detail page for a course
Basic course of events	Write a search words and presses the search button. A list of courses matching the query is displayed and the user can click on the course code on one of them to get to a detail page.
Preconditons	None
Postconditions	None

Browse courses

Goal	Find detail page for a course
Basic course of events	Enter the browse course page by clicking its button in the menu. You can select zero or more of the following varaibles: faculty, institute and semester. A list of courses matching the variables is listed and you can click on the course code on one of them to get to a detail page.
Preconditons	None
Postconditions	None

View all evaluations for a course	
Goal	Display all evaluations for `[U+FFFD]`urse and rate them on various attributes.
Basic course of events	The user views an evaluation somewhere on the site. If the evaluation is useful, the user click on 'useful', if not, click on 'not useful'.
Preconditons	The user has not rated that evaluation before
Postconditions	The user cannot rate this evaluation more, and the thumb s/he chose is visible with bold text.

Rate an evaluation	
Goal	Give a thumb up or down on a evaluation
Basic course of events	The user views an evaluation somewhere on the site. If the evaluation is useful, the user click on 'useful', if not, click on 'not useful'.
Preconditons	The user has not rated that evaluation before
Postconditions	The user cannot rate this evaluation more, and the thumb s/he chose is visible with bold text.

Register new user	
Goal	Be able to log in with a username and password
Basic course of events	Enter the form by clicking the 'new user' link at the top of the page. Fill in username from UiO, a e-mail and a password and click the 'create user' button. If the password matches and the e-mail is valid a e-mail is sent to the users UiO-email with a activation link for the new user.
Preconditons	The UiO username must exist, or the email will not be sent. No username check against UiOs systems is done.
Postconditions	None

Log in	
Goal	Get the extra functionality of a logged in user.
Basic course of events	Enter the log in form by clicking the 'log in' link at the top of the page, or try to enter a page (eg. new evaluation) that requires to be logged in. Enter username and password and click log in.
Preconditons	The user has a valid username and password.
Postconditions	The user is in a logged in state and are sent to the profile page. If the user entered the 'log in' page by trying to access log in required content, the user is sent back to that resource.

Request new password	
Goal	Get a new password by e-mail to log in to the site.
Basic course of events	Enter the form from a link on the 'log in' site. Enter the e-mail address you registered with and a new password is sent to it.
Preconditons	The user has a user account and remembers which e-mail address s/he registered with.
Postconditions	None

Logged in

Logged in users naturally have the same functionality as those not logged in. In addition, they can contribute with evaluations, edit their profile and give thumbs up/down on others evaluations. The following tables describe the use cases for logged in users:

Add a semester (accomplishment of a course)	
Goal	Add a semester that in a later stage can be linked to a new evaluation.
Basic course of events	Find the course as described in the above use cases. 'Search course' or 'Browse course'. At the end of the information table at top you will find a row with the header 'Completed'. In its right cell there is a link called 'Add new semester'. Click this and chose the semester you want. You go 5 up to 5 years back in time. Users can also add semester from the 'New evaluation' page.
Preconditons	User must be logged in.
Postconditions	The semester is added and displayed in the right cell and in the user profile page.

Delete a semester	
Goal	Delete the semester and all evaluations linked to it.
Basic course of events	Enter the profilepage from the link called 'Profile page' at the top right of the site. A list of all your course semester is displayed under 'Your courses'. Find the semester you want to delete, and click the red cross icon at the right of the row.This will also trigger the deletion of all evaluations linked to the semester. This is written in the confirmation box.
Preconditons	The user must be logged in.
Postconditions	The semester and all its evaluations is removed from all parts of the system.

Write evaluation	
Goal	Write and publish an anonymous evaluation
Basic course of events	Find the course as described in the above use cases 'Search course' or 'Browse course'. To enter the evaluation form, click the link 'Write a new evaluation'. All form elements, except 'Tutorials / Seminars' are mandatory. A semester must be chosen, if the user does not have this, s/he can add it by clicking the link 'Add new semester'. The user can add up to three semesters pr. course. The evaluation form link can also be reached from some other pages within the site.
Preconditons	User must be logged in.
Postconditions	The evaluation is published and visible to all users.

Delete an evaluation.	
Goal	Delete the evaluation.

Basic course of events	Enter the profilepage from the link called 'Profile page' at the top right of the site. In the tab menu at the top, click 'All your evaluations'. Find the evaluation you want to delete and click the link 'Delete evaluation' at the bottom right of the evaluation box. This will present a confirmation box, click OK, and it will be deleted.
Preconditons	The user is logged in.
Postconditions	The evaluation is deleted from all parts of the system.

Change personal information	
Goal	Change e-mail address, institute and faculty.
Basic course of events	Enter the profilepage from the link called 'Profile page' at the top right of the site. Enter the form by clicking the 'new user' link at the top of the page. Expand the personal information section by clicking the header 'Your information'. Fill in the new information and click the 'Save' button.
Preconditons	The user must be logged in.
Postconditions	The new personal information is stored in the database.

Change password	
Goal	Change the password to a user account.
Basic course of events	Enter the profilepage from the link called 'Profile page' at the top right of the site. In the tab menu at the top, click the 'Change password' link. Fill in the old password followed by the new password twice. Click the button 'Save password'.
Preconditons	The user is logged in.
Postconditions	The password is changed in the database.

Log out	
Goal	Log out and clear the session.
Basic course of events	Click the log out button at the top right of the page.
Preconditons	The user is logged in.

Postconditions	The user is logged out of the site.

A.3.2 back-end

For a front-end of a web application to work well, we need a good and solid foundation. Kurskritikk will use Django (more in 4.4.1 on page 29) as a web framework. This is an open source Python based framework which provides good security, extendibility and easy maintenance. As a database, MySQL will be used. This is also open-source, and the most popular of them.

To keep the users active, Kurskritikk needs course information that is up to date. With UiO this is made possible via XML documents that are updated every semester. On the site http://www.uio.no/vrtx/mapping/, UiO provides XML documents for all courses at UiO.

A.3.3 Security

Kurskritikk's security will be dependent on Django and its implementation. As an open source framework with a big user community any potential security issues most often will be reported early. Django never publishes the security issues before they have a fix, but as an open source project, anyone can get access to them if they want. This will always be a problem, and we just have to keep up the pace and stay in front of them.

SQL injection

Django protects the web application through a couple of built in features. SQL injection will not be possible since the database API automatically escapes all SQL parameters before querying the database. As a default the web application should never trust data from users no matter what.

Cross Site Scripting

Another known web vulnerability that we need protection against is Cross Site Scripting (XSS). This is most often exploited when user submitted data is rendered into HTML. This can be handled in Django by always escaping HTML characters from the evaluations and other user input.

Session Forging/Hijacking

This is a class of attacks that include man-in-the-middle, session forging, cookie forging, session fixation, session poisoning. All of these are attacks on the user's session data. This should be prevented by

- Never allow session data to be contained in the URL.
- Do not store data in cookies directly, but via a session ID that maps to session data.
- Escape session data if it is displayed in a template.
- Prevent attackers from spoofing session IDs whenever possible.

Email header injection

Kurskritikk will in its first version not have any email forms. If in the future it will, any headers in emails should be escaped before sending data. Djangos built-in mail functions do this.

Exposed error messages

Error messages are very useful when developing and debugging a web application. Django provides a debug flag that if set true, it will show aspects of the code and configuration that can aid an attacker. Such error messages will most often never be at any use for end users, so the main philosophy should be to always show friendly error messages in a live web application.

Password encryption

User passwords must be encrypted with a one way hash of the password. Sha1 is Djangos default and is used in its built-in password functions. This will prevent anyone but the user to get access to the password. If a user forgets the password, a new must be generated and sent to him/her.

A.3.4 Fetching XML data from UiO

All course information at the University of Oslo is available in XML-format. This system will take advantage of this, fetch the information and use it to offer an updated database of courses. In the first version of Kurskritikk, a script that reads the XML and updates the database will suffice. This script can be executed manually twice a year, and in a future version automatically twice a year. It must get the course data

- Name
- Course code
- Faculty
- Institute
- Credits
- URL

In a future version this script can be extended to also fetch the course description.

A.3.5 Database

For database I will use MySQL 5, which is licensed under the GPL license. It has proven to be the best open source database on the market, and seems to be a good fit for this site. It is extendible, fast and has support in Django.

A.3.6 Quality

The application should be easy to maintain and have good reusability and scalability. It should be easy to implement user configuration such as internationalization and personalization. Kurskritikk must be a reliable, robust service with low response time.

A.4 Non-functional requirements

The site must have an appealing look and be easy to navigate. This infers a good structure on all data which is presented. The site should have consistent use of key words through all functions and services.

A.4.1 Project drivers

After almost 5 years at the University of Oslo, I have written a couple of course evaluations. The input focus on the quality and amount of the curriculum, tutorial classes and lectures. Their focus is to help the course administrations to improve the overall quality of the course in time.

What drove me into the idea of making this site was the potential in all the information the students give away in these evaluations. If we took this information, opened it up for the public and gave each student the possibility to connect this information to their course lists, it would open up a sea of possibilities.

A.4.2 User groups

Course administrations

The goal of this site is not only to open up the existing course administration for everyone, but the intention is also to increase the participation rate. With students getting value for their effort straight away, hopefully more of them will contribute. This will then lead to more valuable information for the course administrations. With an open system like this, the administrations will have to really follow up on critics. Let's say one course gets bad evaluations all the way. Bad lecturer, bad curriculum and bad tutorials. This will now be much more visible for the students. Where these bad 'rumours' earlier went through several students through oral channels, students now have direct access to them. In time, a site like this may contribute to easier course choices for students and quality assurance from administrations.

Students

Not much that have not been covered yet, but let's take a quick go-through. The site will help students browse and also choose courses based on hands on information from fellow students who have taken the same courses. Today's official information on web can be somewhat abstract and does not always discuss how the course is practically. It does not discuss at all how the lecturers are and what students think of the course. All this is important and today missing information for students who are insecure of which courses they want to take.

All this information can be useful for both existing UiO-students and students in other universities across the country.

Appendix B

System documentation

B.1 Introduction

Kurskritikk is a result of several months of development. Code is written in Python (using Django), and HTML through Django templates which is a mix of HTML and Python. All graphics is done in GIMP[1], and the HTML styling in CSS[2]. The following list gives an overview of the applications size:

- **Lines of Python:** 2067
- **Lines of templates:** 2243
- **Lines of CSS:** 900

This system documentation will now describe how Kurskritikk uses Django and other technologies to rapidly build a good, stable and functional web application. I will not describe the whole application in the system documentation, but extract the most central and relevant parts. No parts of the design (HTML, CSS and graphics) will be discussed here. All source code and graphics can be found in the attached CD and run locally. You can also test the application online with these credentials:

- **URL:** http://www.kurskritikk.no/
- **Username:** testuser
- **Password:** usertest

In order to understand how all parts of Kurskritikk is made, I encourage the reader to get familiar with Djangos documentation and then read the source code. The source code, its structure and how to run Kurskritikk on a local machine is described in section C.1.2 on page 66

[1] GNU Image Manipulation Program - www.gimp.org
[2] Cascading Style Sheets

B.2 Python

Kurskritikk is developed using the Python framework Django. Python is a multi-paradigm programming language with garbage collection. Wikipedia defines Python like this:

> Python is a general-purpose, very high-level programming language. Its design philosophy emphasizes programmer productivity and code readability. Python's core syntax and semantics is minimalist, while the standard library is large and comprehensive.
>
> Python supports multiple programming paradigms (primarily object oriented, imperative, and functional) and features a fully dynamic type system and automatic memory management; similar to Perl, Ruby, Scheme, and TCL.

The object orientation is strong in Python, all data in Python are objects. The keywords in Python are simple and similar to other programming languages. The following identifiers are used as reserved words, or keywords of the language:

Python keywords				
and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

Python have the number types plain `integer, long integer, floating point numbers` and `complex numbers`. The type system is dynamic which means no keyword is needed to define variables, Python automatically determines if it's a number or a sequence type. The sequence types are `strings, Unicode strings, lists, tuples, buffers`, and `xrange objects`. In Kurskritikk unicode strings and lists are mostly used. Mapping types is the last and most used. In Python there is only one of these, the dictionary. A dict looks like this: `{key: value}` where the key could be a number or string, and the value can be anything. It is normal to have a list or another dict as the value.

Casting is possible through casting functions. Examples of these are `int(), float()` and `str()` where you send the casting object as a parameter to the function. A unique Python feature is that the indention is parts of the languages semantics. You do not have { ...} curly brackets around classes, loops or functions. A simple : after the header is enough, and the all code is then forced to be correctly indented, or it will not run. Semi colons are not a part of the semantics as we are used to in other languages like Java and C. The strict indention syntax makes the ; unnecessary.

B.3 Django in Kurskritikk

Django is as they say in their site a "*high-level Python Web framework that encourages rapid development and clean, pragmatic design*". It is described in more detail in section 4.4.1 on page 29.

B.3.1 Database and Djangos object relational mapper

The foundation of a Django web application is the models. They provide an access to the applications database through an object relational mapper (ORM). The models are written as a class for each table in the DB. When you are finished with the models, Django provides the synchronization script `syncdb` which builds this database and automatically assigns primary keys to the tables if desirable? These classes can then be accessed via an extensive database API. The models are tightly connected to the built-in administration interface Django offers. In each model you can have an inner class which tells which fields should be visible in the admin interface you also define a `__unicode__` function that returns a human readable name for the object when referenced in the admin interface. To see what it really looks like, I have drawn an entity relational diagram (B.1 on page 59) and pasted parts of the `models.py` file showing the classes `Faculty, UserProfile, Course, UserCourse, Evaluation and UserfulScore`. First I want to give a quick overview to all the tables/classmodels in Kurskritikk.

Model	**Function**
Faculty	Stores a faculty.
Institute	Stores a institute which has a foreign key to faculty.
User	This is a built-in model in Django which provides the most common used attributes such as username, name and email.
UserProfile	This is a application made model which is related to User with a one-to-one relationship and extends the user with any additional information the application will need.
Semester	Just stores 3 semester values. 1 for spring, 2 for fall, 3 for both.
Course	Stores a course.
UserCourse	Stores a accomplishment of a course, called a semester of a course in the application. This is the model all evaluations are related to.
CourseVisit	A model for statistics. Stores every visit to a course page.
Evaluation	Stores an evaluation which is related to UserCourse and a User.

UsefulScore	Stores a usefulscore (thumbs up) on a evaluation. If the user is logged in at the moment of rating, the user will be stored, if not it will be attached to a session in the users browser.
NotUsefulScore	The same as UsefulScore, but stores thumbs down.
Stats	A single unrelated model which stores some key statistics used in the sorting algorithm.

Table B.2: The models (tables) in Kurskritikk

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

class Faculty(models.Model):
    name = models.CharField(max_length=100)
    url = models.URLField(max_length=100)
    code = models.CharField(max_length=100, unique=True)
    visible = models.BooleanField(blank=True)

    def __unicode__(self):
        return self.name

    class Admin:
        list_display = ('name','url', 'code')
        list_filter = ('name', 'code')

class UserProfile(models.Model):
    user = models.ForeignKey(User, unique=True)
    faculty = models.ForeignKey(Faculty, null=True, blank=True)
    institute = models.ForeignKey(Institute, null=True, blank=True)
    activation_key = models.CharField(max_length=40)
    key_expires = models.DateTimeField()

    class Admin:
        list_display = ('user', 'faculty', 'institute')

class Course(models.Model):
    faculty = models.ForeignKey(Faculty, null=True)
    institute = models.ForeignKey(Institute, null=True)
    course_code = models.CharField(max_length=20)
    name = models.CharField(max_length=200)
    credits = models.IntegerField()
    semester = models.ForeignKey(Semester)
    url = models.URLField()
    average_score = models.FloatField(null=True, blank=True)

    def __unicode__(self):
        return '%s: %s - %s' % (self.id, self.course_code, self.name)

    class Admin:
        list_display = ('course_code', 'name', 'institute', 'faculty', '
```

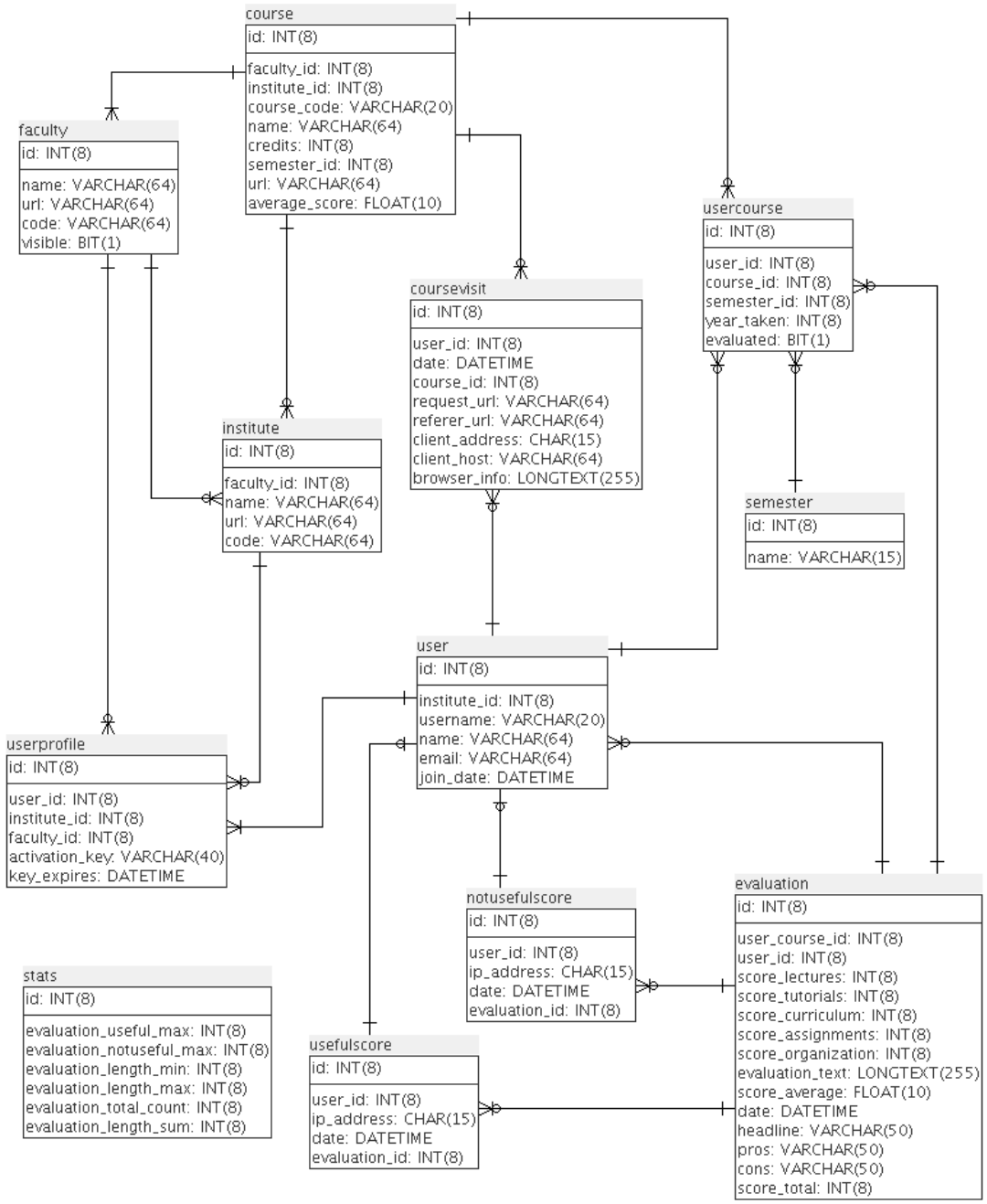

Figure B.1: Kurskritikk's database design

```
            semester', 'average_score')

class UserCourse(models.Model):
    user = models.ForeignKey(User)
    course = models.ForeignKey(Course)
    semester = models.ForeignKey(Semester)
    year_taken = models.IntegerField()
    evaluated = models.BooleanField(default=False)

    def __unicode__(self):
        return '%s: %s - %s' % (self.id, self.user, self.course.
            course_code)

    class Admin:
        list_display = ('id', 'user', 'course', 'semester', 'year_taken',
            'evaluated')

class Evaluation(models.Model):
    user_course = models.ForeignKey(UserCourse, null=True)
    user = models.ForeignKey(User)

    score_lectures = models.IntegerField()
    score_tutorials = models.IntegerField()
    score_curriculum = models.IntegerField()
    score_assignments = models.IntegerField()
    score_organization = models.IntegerField()
    score_total = models.IntegerField()
    evaluation_text = models.TextField()
    headline = models.CharField(max_length=50, null=True, blank=True)
    pros = models.CharField(max_length=50, null=True, blank=True)
    cons = models.CharField(max_length=50, null=True, blank=True)
    score_average = models.FloatField(null=True)
    date = models.DateTimeField(auto_now_add=True)

    def __unicode__(self):
        return '%s: %s - %s' % (self.id, self.user, self.user_course.
            course.course_code)

    class Admin:
        list_display = ('id', 'user_course', 'user', 'score_lectures',
                        'score_tutorials', 'score_curriculum', '
                            score_assignments',
                        'score_organization', 'score_total',)



class UsefulScore(models.Model):
    user = models.ForeignKey(User, null=True)
    ip_address = models.IPAddressField()
    date = models.DateTimeField(auto_now_add=True)
    evaluation = models.ForeignKey(Evaluation)

    def __unicode__(self):
        return '%s: %s' % (self.id, self.evaluation)

    class Admin:
        list_display = ('user', 'ip_address', 'date', 'evaluation')
```

Listing B.1: Model for the table Course

As we can see the database is easy to design through Django writing just normal Python code. Now it is time to have look at how the rest of Django works. I will present the next section in the order an end user would experience it from entering a page till it is displayed.

B.3.2 URLs and views

Django has a URL definition file called `urls.py`. The URLs listed in that file is linked to views that are called when a URL typed in by a uses matches that in a definition. These views are the C in MVC, the control or business logic.

URLs

When a user enters a URL, the first thing Django does is to do a lookup in `urls.py` file in Django is where you build the URL structure of the site. This is a great feature of Django which gives good overview of the site and makes it easy to design useful and nice looking URLs. The URLs are designed using regular expressions (regex) which is a powerful language for string manipulation and search. As I described in section 4.4.1 on page 29, the URL definitions are regexes which matches the tail of the URL, the part after the domain name. In `www.kurskritikk.no/kurs/INF5270/alle_evalueriinger/side1/` this part would be `kurs/INF5270/alle_evalueriinger/side1/`. The URL definitions are built like this:

```
(r'[regular expression]', '[view]', {optional template})
```

When the URL matches the regex, the view is called. Inside the regex, we can have variables which basically are parts of the URLs, matched like this:

```
(?P<course_code>[\w\-]{3,15})
```

This matches all word characters (letters and digits) and hyphens with a length from 3 to 15 characters. This is passed as an argument to the view. A selection of some of Kurskritikk's central `urls.py` can be viewed in (listing B.2).

```
1  urlpatterns = patterns('kurskritikk.kkweb.views',
2      (r'^$', 'index'),
3      (r'^kurs/(?P<course_code>[\w\-]{3,15})/$', 'view_course'),
4      (r'^kurs/(?P<course_code>[\w\-]{3,15})/ny_evaluering/$', '
           new_evaluation'),
5      (r'^kurs/(?P<course_code>[\w\-]{3,15})/evaluering/(?P<id>[0-9]+)+/$',
            'view_evaluation'),
6      (r'^konto/ny_bruker/$', 'ny_bruker'),
7      (r'^konto/logg_inn/$', login, {'template_name': 'logg_inn.html'}),
8      (r'^konto/profil/$', 'profile'),
9      (r'^konto/profil/bekreft/(?P<activation_key>.*)/$', 'confirm'),
10     (r'^konto/profil/endre_passord/', 'change_password'),
```

```
11      (r'^konto/glemt_passord/', 'forgot_password'),
12      (r'^konto/delete_evaluation/', 'delete_evaluation'),
13      (r'^add_course/$', 'add_course'),
14      (r'^save_profile/$', 'save_profile'),
15      (r'^add_useful_score/$', 'add_useful_score'),
16      (r'^sok/(?P<search_string>.+)/(?P<sorting>[\w\_]+)/side(?P<page
            >[0-9]+)/$', 'search'),
17      (r'^statistikk/$', 'statistics'),
18  )
```

Listing B.2: Kurskritikks URL structure

Views

The views are Kurskritikk's largest part in lines of code. All together they consist of more than 1200 lines. In order to show how Kurskritikk uses Django, I will describe how one of them works. `view_evaluation` is the view that is called when a user visits the e.g. this URL: `www.kurskritikk.no/kurs/INF1000/evaluering/75/`.

```
1  def view_evaluation(request, course_code, id):
2    e = get_object_or_404(Evaluation, id=id)
3    score_range = [1, 2, 3, 4, 5]
4
5    e.has_rated = has_rated_evaluation(request, e)
6    e.rating_img = get_rating_img(e.score_total, 'grey_')
7
8    c = Course.objects.get(pk = e.user_course.course.id)
9    evaluation_count = Evaluation.objects.filter(
10             user_course__course = c).count()
11
12   return render_to_response(
13     'view_evaluation.html',
14     {'course_evaluation_count': evaluation_count,
15     'evaluation': e, 'score_range': score_range},
16     context_instance=RequestContext(request))
```

Listing B.3: View: view_evaluation

In the function declaration we see that `view_evaluation` gets three arguments, the request object (all view functions get this), the course code and the evaluation id. The two last ones are sent from the `urls.py`. Then one of Djangos most elegant DB API-call is made, the `get_object_or_404` looks for evaluations in the model `Evaluation` with id equal to the argument `id`. If it does not exist, a 404 exception is raised and the 404 template called. Two helping functions are called, `has_rated_evaluation` checks if the user has rated this evaluation (does a lookup in the users session), and `get_rating_img` gets the graphics (red stars) for the evaluation. If it does not yet have any evaluations, an empty string is returned and no rating displayed in the template. Then it looks up the course and the number of evaluations written on the course. All this is then sent to the template `view_evaluation.html` which displays the evaluation to the user.

B.3.3 Templates

Django has developed a template language which is similar to Smarty or CheetahTemplate. They are called from the views, and sometimes directly from the URL definitions. They are organized in a hierarchical way where a base template is at the top, and other templates are sub templates of the base template. In these templates we can display data from the incoming arguments. These can be either simple variables such as integers and strings, or more complex ones such as a list or a dictionary. Django offers some tag and filter references that does something with the data before displaying it. In the front page of Kurskritikk I use one of these to display only the first 50 characters of an evaluation.

B.3.4 Some other parts of Django and the application

Django has a lot more functionality than I need or should explain in this thesis. To wrap in this chapter I just want to quickly look at some of Djangos other features and a couple of manually made helper functions. In addition to the views I have already described, Django has some built-in ones. In Kurskritikk I use one for log in and log out where the input form is made and business logic is done for you. Another built-in feature Kurskritikk takes use of is the forms library. With this generation and validation of forms with data connected to the models are made really simple.

Middleware is another neat feature of Django. It is components that can handle data before a view does, and simply controls some of the things we often take for granted that happens in the background. Some examples are `auth` which handles the current logged in user, `session` which enables session support in browsers and `flatpages` which makes it easier to create and edit flat pages such as 'about', 'contact' and so on.

In addition to all Django-specific code, I have written some helper scripts to support the main application. I will not go into any depth on these, but just mention them. First of all we have a script that reads the XML documents from UiO and adds or updates course information in the database. To take some load of the views I have made some helper functions to support them. The most important one is `eval_quality` which is used to give an evaluation a rating from 0-1 based on the quality algorithm made in this thesis. Others include one for getting a score image for an evaluation (the red stars) and checking if a user has rated an evaluation.

Appendix C

Attachments

C.1 The source CD

C.1.1 The source structure

The following table shows the structure of the source code and describes the relevant directories and files.

Folder / File	**Description**
kurskritikk/	Django project folder
kurskritikk/kkweb/	Django application folder
kurskritikk/sql/	Backup of some database tables. Used by manage.py to fill the database when it is built from scratch.
kurskritikk/kkweb/templates/	All templates, which mainly are called from the views.
kurskritikk/kkweb/templates/snippets/	Small parts of template code that is reused in more than one template.
kurskritikk/kkweb/extra_functions.py	All help functions that some views are using.
kurskritikk/kkweb/kkforms.py	All forms that are used in the views. Examples are the evaluation form and user registration form.
kurskritikk/kkweb/models.py	The models which is the foundataion for the database and the built-in object relational mapper.
kurskritikk/kkweb/urls.py	All URL definitions, as explained in appendix B.

kurskritikk/kkweb/views.py	The business logic of the application. The largest file with approximately 1200 lines of code. These views are called from the URL definitions.
kurskritikk/templates/	Templates for all flat pages (about, contact etc) and the base template for the project.
kurskritikk/manage.py	A script containing some help functions like model sync against database and a local development server.
kurskritikk/settings.py	All settings for the Kurskritikk application such as database information, path to media and some other constants used by the application.
kurskritikk/media/	All CSS, images and JavaScript.
kurskritikk/scripts/kursmapping.py	The script used to build the course database. It reads a XML document from UiO.

Table C.1: The source code structure

C.1.2 Running Kurskritikk on a local test server

In order to run Kurskritikk on a local machine you will need Django and Python installed, and an Internet connection. You will also need to change some attributes in `settings.py`. The following steps will lead to a running web application:

1. Download and install Django from http://www.djangoproject.com/download/
2. Download and install Python from http://python.org/download/
3. Copy the Kurskritikk folder from the CD to your hard drive.
4. Change the following attributes in the file `kurskritikk/settings.py`.
   - `MEDIA_ROOT`: path to `kurskritikk/media/`.
   - `TEMPLATE_DIRS`: path to `kurskritikk/templates/`.
   - `EMAIL_HOST`: smtp server from the internet provider you are using.
5. In Windows click start > run and type cmd and press enter. In Linux open a terminal window.
6. Enter the kurskritikk folder
7. Type in: `python manage.py runserver 7000.`
8. Open your browser (preferrably not IE 6 but a new version of any browser)

9. Open the URL http://127.0.0.1:7000

The local settings file has debugging set to true, so that if any error occurs, thorough and helpful error messages will be displayed. This can be set in the settings constant `DEBUG`.

C.2 All evaluations sorted on quality

Position	Lenght	TU	TD	Score
1	1981	4	0	0.939
2	1864	2	0	0.929
3	1477	16	2	0.913
4	1330	6	0	0.911
5	1423	3	0	0.905
6	1339	4	0	0.902
7	1286	2	0	0.886
8	930	9	0	0.885
9	1226	1	0	0.875
10	832	8	0	0.87
11	1130	1	0	0.865
12	1599	6	1	0.864
13	946	3	0	0.856
14	726	6	0	0.849
15	967	1	0	0.848
16	802	4	0	0.847
17	733	3	0	0.834
18	822	1	0	0.833
19	706	3	0	0.832
20	332	10	0	0.795
21	418	9	0	0.789
22	323	4	0	0.764
23	364	4	0	0.764
24	378	3	0	0.759
25	532	3	0	0.759
26	358	3	0	0.759
27	442	3	0	0.759
28	512	3	0	0.759
29	430	3	0	0.759
30	343	2	0	0.754

31	643	2	0	0.754
32	470	2	0	0.754
33	286	2	0	0.754
34	464	2	0	0.754
35	511	2	0	0.754
36	240	3	0	0.752
37	291	1	0	0.749
38	322	1	0	0.749
39	389	1	0	0.749
40	505	1	0	0.749
41	604	1	0	0.749
42	230	2	0	0.74
43	222	2	0	0.734
44	214	2	0	0.729
45	214	1	0	0.724
46	211	1	0	0.722
47	175	2	0	0.701
48	161	3	0	0.697
49	161	3	0	0.697
50	159	2	0	0.69
51	136	2	0	0.674
52	137	1	0	0.67
53	114	1	0	0.654
54	220	3	1	0.611
55	251	2	1	0.586
56	777	0	0	0.58
57	524	3	2	0.554
58	203	2	1	0.553
59	416	3	3	0.5
60	555	2	2	0.5
61	425	0	0	0.5
62	225	0	0	0.483
63	170	1	1	0.444
64	125	1	1	0.412
65	65	2	4	0.279
66	396	0	1	0.251
67	315	0	1	0.251
68	331	0	1	0.251

69	484	0	1	0.251
70	52	0	1	0.113
71	98	1	16	0.102

Table C.2: All evaluations sorted on quality score

Bibliography

[1] J. M. Barbalet. A social emotions: Confidence, trust and loyalty. *International Journal of Sociology and Social Policy*, pages 75–96, 1996.

[2] R. Clarke. Identification, anonymity and pseudonymity in consumer transactions: a vital systems design and public policy issue. *Proc. Conf. "Smart Cards: The Issues"*, 1996.

[3] E. Davison and J. Price. How do we rate? an evaluation of online student evaluations. *Unpublished manuscript*, 2006.

[4] C. Diaz, S. Seys, J. Claessens, and B. Preneel. Towards measuring anonymity. pages 54–68. Springer-Verlag, Berlin, Heidelberg, 2003.

[5] R. Guha. Cv. http://www.guha.com/cv.html.

[6] R. Guha. Open rating systems. *Proc. 1st Workshop on Friend of a Friend*, 2004.

[7] IBM. Second life press release. http://www-03.ibm.com/press/us/en/pressrelease/21551.wss.

[8] Privacy International. Overview of privazy. http://www.privacyinternational.org/article.shtml?cmd[347]=x-347-559062.

[9] T. Joachims. Making large-scale svm learing practical. *Advances in Kernel Methods: Support Vector Learning*, 1999.

[10] Peter G. Kilner and Christopher M. Hoadley. Anonymity options and professional participation in an online community of practice. In *CSCL '05: Proceedings of th 2005 conference on Computer support for collaborative learning*, pages 272–280. International Society of the Learning Sciences, 2005.

[11] Dekang Lin. Making large-scale svm learing practicalprincipar - an efficient, broad-coverage, principle-based parser. In *COLING*, pages 42–48, Tokyo, Japan, 1994.

[12] B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *Proceedings of the ACL*, 2005.

[13] Alex Patriquin. Connecting the social graph: Member overlap at opensocial and facebook.

[14] T. Chklovski M. Pennacchiotti Soo-Min Kim, P. Pantel. Automatically assessing review helpfulness. *EMNLP 2006*, pages 423–430, 2006.

[15] C. Spearman. The proof and measurement of accosiation between two things. *American Journal of Psychology*, pages 15:72–101, 1904.

[16] P. Sztompka. *Trust: A Sociological Theory*. Cambridge University Press, 1999.

[17] Y. Wang and Y. Vassileva. Toward trust and reputation based web service selection: A survey. *Multi-agent and Grid Systems (MAGS)*, 2007.

[18] Wikipedia. Social network service. http://en.wikipedia.org/wiki/Social_network_service/.

[19] H. Yu and V. Hatzivassiloglou. Towards answering opinion questions: Separating facts from opinions and identifying the polarity of opinion sentences. *Proceedings of EMNLP*, 2003.